

Francesco Camastra  
Alessandro Vinciarelli

# Machine Learning for Audio, Image and Video Analysis

SPIN Springer's internal project number

October 5, 2007

Springer



To our parents and families



---

# Contents

|          |  |   |
|----------|--|---|
| <b>1</b> | <b>Introduction</b> .....                    | 1 |
| 1.1      | Two Fundamental Questions .....              | 1 |
| 1.1.1    | Why Should One Read The Book? .....          | 1 |
| 1.1.2    | What Is the Book About? .....                | 2 |
| 1.2      | The Structure of the Book .....              | 4 |
| 1.2.1    | Part I: From Perception to Computation ..... | 4 |
| 1.2.2    | Part II: Machine Learning .....              | 5 |
| 1.2.3    | Part III: Applications .....                 | 6 |
| 1.2.4    | Appendices .....                             | 7 |
| 1.3      | How to Read This Book .....                  | 7 |
| 1.3.1    | Background and Learning Objectives .....     | 8 |
| 1.3.2    | Difficulty Level .....                       | 8 |
| 1.3.3    | Problems .....                               | 8 |
| 1.3.4    | Software .....                               | 9 |
| 1.4      | Reading Tracks .....                         | 9 |

---

## Part I From Perception to Computation

---

|          |  |    |
|----------|--|----|
| <b>2</b> | <b>Audio Acquisition, Representation and Storage</b> ..... | 13 |
| 2.1      | Introduction .....   | 13 |
| 2.2      | Sound Physics, Production and Perception .....             | 15 |
| 2.2.1    | Acoustic Waves Physics .....                               | 15 |
| 2.2.2    | Speech Production .....                                    | 18 |
| 2.2.3    | Sound Perception .....                                     | 20 |
| 2.3      | Audio Acquisition .....                                    | 22 |
| 2.3.1    | Sampling and Aliasing .....                                | 23 |
| 2.3.2    | The Sampling Theorem** .....                               | 25 |
| 2.3.3    | Linear Quantization .....                                  | 27 |
| 2.3.4    | Nonuniform Scalar Quantization .....                       | 30 |
| 2.4      | Audio Encoding and Storage Formats .....                   | 32 |

|                         |  |           |
|-------------------------|--|-----------|
| 2.4.1                   | Linear PCM and Compact Discs .....                               | 32        |
| 2.4.2                   | MPEG Digital Audio Coding .....                                  | 34        |
| 2.4.3                   | AAC Digital Audio Coding .....                                   | 35        |
| 2.4.4                   | Perceptual Coding .....  | 36        |
| 2.5                     | Time-Domain Audio Processing .....                               | 38        |
| 2.5.1                   | Linear and Time-Invariant Systems .....                          | 38        |
| 2.5.2                   | Short-Term Analysis .....  | 40        |
| 2.5.3                   | Time-Domain Measures .....                                       | 41        |
|                         | Problems .....   | 46        |
| <b>References .....</b> |  | <b>49</b> |
| <b>3</b>                | <b>Image and Video Acquisition, Representation and Storage .</b> | <b>51</b> |
| 3.1                     | Introduction .....   | 51        |
| 3.2                     | Human Eye Physiology .....                                       | 52        |
| 3.2.1                   | Structure of the Human Eye .....                                 | 52        |
| 3.3                     | Image Acquisition Devices .....                                  | 54        |
| 3.3.1                   | Digital Camera .....   | 54        |
| 3.4                     | Color Representation .....                                       | 57        |
| 3.4.1                   | Human Color Perception .....                                     | 57        |
| 3.4.2                   | Color Models .....   | 59        |
| 3.5                     | Image Formats .....  | 66        |
| 3.5.1                   | Image File Format Standards .....                                | 66        |
| 3.5.2                   | JPEG Standard .....  | 68        |
| 3.6                     | Video Principles .....   | 72        |
| 3.7                     | MPEG Standard .....  | 73        |
| 3.7.1                   | Further MPEG Standards .....                                     | 75        |
| 3.8                     | Conclusions .....  | 77        |
|                         | Problems .....   | 78        |
| <b>References .....</b> |  | <b>79</b> |

---

## Part II Machine Learning

---

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>4</b> | <b>Machine Learning .....</b>      | <b>83</b> |
| 4.1      | Introduction .....                 | 83        |
| 4.2      | Taxonomy of Machine Learning ..... | 84        |
| 4.2.1    | Rote Learning .....                | 84        |
| 4.2.2    | Learning from Instruction .....    | 85        |
| 4.2.3    | Learning by Analogy .....          | 85        |
| 4.3      | Learning from Examples .....       | 85        |
| 4.3.1    | Supervised Learning .....          | 86        |
| 4.3.2    | Reinforcement Learning .....       | 86        |
| 4.3.3    | Unsupervised Learning .....        | 87        |

|          |   |            |
|----------|---|------------|
| 4.4      | Conclusions . . . . .                                       | 88         |
|          | <b>References . . . . .</b>                                 | <b>89</b>  |
| <b>5</b> | <b>Bayesian Theory of Decision . . . . .</b>                | <b>91</b>  |
| 5.1      | Introduction . . . . .                                      | 91         |
| 5.2      | Bayes Decision Rule . . . . .                               | 92         |
| 5.3      | Bayes Classifier* . . . . .                                 | 95         |
| 5.4      | Loss Function . . . . .                                     | 96         |
| 5.4.1    | Binary Classification . . . . .                             | 98         |
| 5.5      | Zero-One Loss Function . . . . .                            | 99         |
| 5.6      | Discriminant Functions . . . . .                            | 100        |
| 5.6.1    | Binary Classification Case . . . . .                        | 101        |
| 5.7      | Gaussian Density . . . . .                                  | 101        |
| 5.7.1    | Univariate Gaussian Density . . . . .                       | 102        |
| 5.7.2    | Multivariate Gaussian Density . . . . .                     | 102        |
| 5.7.3    | Whitening Transformation . . . . .                          | 104        |
| 5.8      | Discriminant Functions for Gaussian Likelihood . . . . .    | 106        |
| 5.8.1    | Features Are Statistically Independent . . . . .            | 106        |
| 5.8.2    | Covariance Matrix Is The Same for All Classes . . . . .     | 107        |
| 5.8.3    | Covariance Matrix Is Not the Same for All Classes . . . . . | 109        |
| 5.9      | Receiver Operating Curves . . . . .                         | 109        |
| 5.10     | Conclusions . . . . .                                       | 111        |
|          | Problems . . . . .  | 112        |
|          | <b>References . . . . .</b>                                 | <b>115</b> |
| <b>6</b> | <b>Clustering Methods . . . . .</b>                         | <b>117</b> |
| 6.1      | Introduction . . . . .                                      | 117        |
| 6.2      | Expectation and Maximization Algorithm* . . . . .           | 119        |
| 6.2.1    | Basic EM* . . . . .   | 120        |
| 6.3      | Basic Notions and Terminology . . . . .                     | 122        |
| 6.3.1    | Codebooks and Codevectors . . . . .                         | 122        |
| 6.3.2    | Quantization Error Minimization . . . . .                   | 124        |
| 6.3.3    | Entropy Maximization . . . . .                              | 124        |
| 6.3.4    | Vector Quantization . . . . .                               | 125        |
| 6.4      | K-Means . . . . .   | 127        |
| 6.4.1    | Batch K-Means . . . . .                                     | 128        |
| 6.4.2    | Online K-Means . . . . .                                    | 129        |
| 6.4.3    | K-Means Software Packages . . . . .                         | 132        |
| 6.5      | Self-Organizing Maps . . . . .                              | 132        |
| 6.5.1    | SOM Software Packages . . . . .                             | 134        |
| 6.5.2    | SOM Drawbacks . . . . .                                     | 134        |
| 6.6      | Neural Gas and Topology Representing Network . . . . .      | 134        |
| 6.6.1    | Neural Gas . . . . .  | 135        |

|          |  |     |
|----------|--|-----|
| 6.6.2    | Topology Representing Network                                  | 135 |
| 6.6.3    | Neural Gas and TRN Software Package                            | 137 |
| 6.6.4    | Neural Gas and TRN Drawbacks                                   | 137 |
| 6.7      | General Topographic Mapping*                                   | 137 |
| 6.7.1    | Latent Variables*  | 137 |
| 6.7.2    | Optimization by EM Algorithm*                                  | 139 |
| 6.7.3    | GTM versus SOM*  | 140 |
| 6.7.4    | GTM Software Package   | 141 |
| 6.8      | Fuzzy Clustering Algorithms                                    | 141 |
| 6.8.1    | FCM  | 142 |
| 6.9      | Hierarchical Clustering  | 142 |
| 6.10     | Conclusion   | 144 |
|          | Problems   | 145 |
|          | <b>References</b>  | 147 |
| <b>7</b> | <b>Foundations of Statistical Learning and Model Selection</b> | 149 |
| 7.1      | Introduction   | 149 |
| 7.2      | Bias-Variance Dilemma  | 150 |
| 7.2.1    | Bias-Variance Dilemma for Regression                           | 150 |
| 7.2.2    | Bias-Variance Decomposition for Classification*                | 151 |
| 7.3      | Model Complexity   | 153 |
| 7.4      | VC Dimension and Structural Risk Minimization                  | 156 |
| 7.5      | Statistical Learning Theory*                                   | 159 |
| 7.5.1    | Vapnik-Chervonenkis Theory                                     | 161 |
| 7.6      | AIC and BIC Criteria   | 163 |
| 7.6.1    | Akaike Information Criterion                                   | 163 |
| 7.6.2    | Bayesian Information Criterion                                 | 164 |
| 7.7      | Minimum Description Length Approach                            | 165 |
| 7.8      | Crossvalidation  | 166 |
| 7.8.1    | Generalized Crossvalidation                                    | 166 |
| 7.9      | Conclusion   | 168 |
|          | Problems   | 168 |
|          | <b>References</b>  | 171 |
| <b>8</b> | <b>Supervised Neural Networks and Ensemble Methods</b>         | 173 |
| 8.1      | Introduction   | 173 |
| 8.2      | Artificial Neural Networks and Neural Computation              | 174 |
| 8.3      | Artificial Neurons   | 175 |
| 8.4      | Connections and Network Architectures                          | 178 |
| 8.5      | Single-Layer Networks  | 180 |



|          |   |     |
|----------|---|-----|
| 8.5.1    | Linear Discriminant Functions and Single-Layer Networks . . . . . | 181 |
| 8.5.2    | Linear Discriminants and the Logistic Sigmoid . . . . .           | 182 |
| 8.5.3    | Generalized Linear Discriminants and the Perceptron . . . . .     | 183 |
| 8.6      | Multilayer Networks . . . . .                                     | 185 |
| 8.6.1    | The Multilayer Perceptron . . . . .                               | 186 |
| 8.7      | Multilayer Networks Training . . . . .                            | 188 |
| 8.7.1    | Error Back-Propagation for Feed-Forwards Networks* . . . . .      | 188 |
| 8.7.2    | Parameter Update: The Error Surface . . . . .                     | 190 |
| 8.7.3    | Parameters Update: The Gradient Descent* . . . . .                | 191 |
| 8.7.4    | The Torch Package . . . . .                                       | 193 |
| 8.8      | Learning Vector Quantization . . . . .                            | 194 |
| 8.8.1    | The <i>LVQ_PAK</i> Software Package . . . . .                     | 196 |
| 8.9      | Ensemble Methods . . . . .  | 197 |
| 8.9.1    | Classifier Diversity and Ensemble Performance* . . . . .          | 197 |
| 8.9.2    | Creating Ensemble of Diverse Classifiers . . . . .                | 200 |
| 8.10     | Conclusions . . . . .   | 204 |
|          | Problems . . . . .  | 204 |
|          | <b>References</b> . . . . .                                       | 207 |
| <b>9</b> | <b>Kernel Methods</b> . . . . .                                   | 211 |
| 9.1      | Introduction . . . . .  | 211 |
| 9.2      | Lagrange Method and Kuhn Tucker Theorem . . . . .                 | 213 |
| 9.2.1    | Lagrange Multipliers Method . . . . .                             | 213 |
| 9.2.2    | Kuhn Tucker Theorem . . . . .                                     | 215 |
| 9.3      | Support Vector Machines for Classification . . . . .              | 216 |
| 9.3.1    | Optimal Hyperplane Algorithm . . . . .                            | 217 |
| 9.3.2    | Support Vector Machine Construction . . . . .                     | 220 |
| 9.3.3    | Algorithmic Approaches to Solve Quadratic Programming . . . . .   | 222 |
| 9.3.4    | Sequential Minimal Optimization . . . . .                         | 223 |
| 9.3.5    | Other Optimization Algorithms . . . . .                           | 225 |
| 9.3.6    | SVM and Regularization Methods* . . . . .                         | 226 |
| 9.4      | Multiclass Support Vector Machines . . . . .                      | 228 |
| 9.4.1    | One-versus-Rest Method . . . . .                                  | 228 |
| 9.4.2    | One-versus-One Method . . . . .                                   | 229 |
| 9.4.3    | Other Methods . . . . .   | 229 |
| 9.5      | Support Vector Machines for Regression . . . . .                  | 229 |
| 9.5.1    | Regression with Quadratic $\epsilon$ -Insensitive Loss . . . . .  | 230 |
| 9.5.2    | Kernel Ridge Regression . . . . .                                 | 233 |
| 9.5.3    | Regression with Linear $\epsilon$ -Insensitive Loss . . . . .     | 235 |
| 9.5.4    | Other Approaches to Support Vector Regression . . . . .           | 236 |
| 9.6      | Gaussian Processes . . . . .                                      | 237 |
| 9.6.1    | Regression with Gaussian Processes . . . . .                      | 238 |

|           |   |     |
|-----------|---|-----|
| 9.7       | Kernel Fisher Discriminant                                  | 239 |
| 9.7.1     | Fisher's Linear Discriminant                                | 239 |
| 9.7.2     | Fisher Discriminant in Feature Space                        | 240 |
| 9.8       | Kernel PCA  | 242 |
| 9.8.1     | Centering in Feature Space                                  | 243 |
| 9.9       | One-Class SVM   | 245 |
| 9.9.1     | One-Class SVM Optimization                                  | 247 |
| 9.10      | Kernel Clustering Methods                                   | 249 |
| 9.10.1    | Kernel K-Means  | 249 |
| 9.10.2    | One-Class SVM Extensions                                    | 251 |
| 9.10.3    | Spectral Clustering   | 252 |
| 9.11      | Software Packages   | 254 |
| 9.12      | Conclusion  | 255 |
|           | Problems  | 256 |
|           | <b>References</b>   | 259 |
| <b>10</b> | <b>Markovian Models for Sequential Data</b>                 | 265 |
| 10.1      | Introduction  | 265 |
| 10.2      | Hidden Markov Models  | 266 |
| 10.2.1    | Emission Probability Functions                              | 269 |
| 10.3      | The Three Problems  | 270 |
| 10.4      | The Likelihood Problem and the Trellis**                    | 271 |
| 10.5      | The Decoding Problem**                                      | 274 |
| 10.6      | The Learning Problem**                                      | 278 |
| 10.6.1    | Parameter Initialization                                    | 279 |
| 10.6.2    | Estimation of the Initial State Probabilities               | 280 |
| 10.6.3    | Estimation of the Transition Probabilities                  | 280 |
| 10.6.4    | Emission Probability Function Parameters Estimation         | 281 |
| 10.7      | HMM Variants  | 284 |
| 10.8      | $N$ -gram Models and Statistical Language Modeling          | 286 |
| 10.8.1    | $N$ -gram Models  | 287 |
| 10.8.2    | The Perplexity  | 287 |
| 10.8.3    | $N$ -grams Parameter Estimation                             | 288 |
| 10.8.4    | The Sparseness Problem and the Language Case                | 289 |
| 10.9      | Discounting and Smoothing Methods for<br>$N$ -gram Models** | 292 |
| 10.9.1    | The Leaving-One-Out Method                                  | 292 |
| 10.9.2    | The Turing Good Estimates                                   | 294 |
| 10.9.3    | Katz's Discounting Model                                    | 295 |
| 10.10     | Building a Language Model with $N$ -grams                   | 296 |
|           | Problems  | 297 |
|           | <b>References</b>   | 301 |

|  |     |
|--|-----|
| <b>11 Feature Extraction Methods and Manifold Learning</b> |     |
| <b>Methods</b> . . . . .                                   | 305 |
| 11.1 Introduction . . . . .                                | 305 |
| 11.2 The Curse of Dimensionality* . . . . .                | 307 |
| 11.3 Data Dimensionality . . . . .                         | 308 |
| 11.3.1 Local Methods . . . . .                             | 308 |
| 11.3.2 Global Methods . . . . .                            | 309 |
| 11.4 Principal Component Analysis . . . . .                | 313 |
| 11.4.1 Nonlinear Principal Component Analysis . . . . .    | 315 |
| 11.5 Independent Component Analysis . . . . .              | 316 |
| 11.5.1 Statistical Independence . . . . .                  | 317 |
| 11.5.2 ICA Estimation . . . . .                            | 318 |
| 11.5.3 ICA by Mutual Information Minimization . . . . .    | 322 |
| 11.5.4 FastICA Algorithm . . . . .                         | 323 |
| 11.6 Multidimensional Scaling Methods . . . . .            | 325 |
| 11.6.1 Sammon's Mapping . . . . .                          | 325 |
| 11.7 Manifold Learning . . . . .                           | 326 |
| 11.7.1 The Manifold Learning Problem . . . . .             | 327 |
| 11.7.2 Isomap . . . . .                                    | 328 |
| 11.7.3 Locally Linear Embedding . . . . .                  | 329 |
| 11.7.4 Laplacian Eigenmaps . . . . .                       | 331 |
| 11.8 Conclusion . . . . .                                  | 333 |
| Problems . . . . .   | 333 |
| <b>References</b> . . . . .                                | 337 |

---

### Part III Applications

---

|  |     |
|--|-----|
| <b>12 Speech and Handwriting Recognition</b> . . . . . | 345 |
| 12.1 Introduction . . . . .                            | 345 |
| 12.2 The General Approach . . . . .                    | 346 |
| 12.3 The Front End . . . . .                           | 349 |
| 12.3.1 The Handwriting Front End . . . . .             | 349 |
| 12.3.2 The Speech Front End . . . . .                  | 351 |
| 12.4 HMM Training . . . . .                            | 353 |
| 12.4.1 Lexicon and Training Set . . . . .              | 353 |
| 12.4.2 Hidden Markov Models Training . . . . .         | 355 |
| 12.5 Recognition and Performance Measures . . . . .    | 356 |
| 12.5.1 Recognition . . . . .                           | 356 |
| 12.5.2 Performance Measurement . . . . .               | 357 |
| 12.6 Recognition Experiments . . . . .                 | 359 |
| 12.6.1 Lexicon Selection . . . . .                     | 360 |
| 12.6.2 $N$ -gram Model Performance . . . . .           | 361 |
| 12.6.3 Cambridge Database Results . . . . .            | 363 |

|   |            |
|---|------------|
| 12.6.4 IAM Database Results . . . . .                                       | 366        |
| 12.7 Speech Recognition Results . . . . .                                   | 367        |
| 12.8 Applications . . . . .   | 368        |
| 12.8.1 Applications of Handwriting Recognition . . . . .                    | 369        |
| 12.8.2 Applications of Speech Recognition . . . . .                         | 371        |
| <b>References . . . . .</b>   | <b>373</b> |
| <br>  |            |
| <b>13 Automatic Face Recognition . . . . .</b>                              | <b>381</b> |
| 13.1 Introduction . . . . .   | 381        |
| 13.2 Face Recognition: General Approach . . . . .                           | 383        |
| 13.3 Face Detection and Localization . . . . .                              | 385        |
| 13.3.1 Face Segmentation and Normalization with <i>TorchVision</i>          | 387        |
| 13.4 Lighting Normalization . . . . .                                       | 387        |
| 13.4.1 Center/Surround Retinex . . . . .                                    | 388        |
| 13.4.2 Gross and Brajovic’s Algorithm . . . . .                             | 389        |
| 13.4.3 Normalization with <i>TorchVision</i> . . . . .                      | 389        |
| 13.5 Feature Extraction . . . . .   | 390        |
| 13.5.1 Holistic Approaches . . . . .  | 390        |
| 13.5.2 Local Approaches . . . . .   | 392        |
| 13.5.3 Feature Extraction with <i>TorchVision</i> . . . . .                 | 393        |
| 13.6 Classification . . . . .   | 397        |
| 13.7 Performance Assessment . . . . .                                       | 399        |
| 13.7.1 The FERET Database . . . . .   | 400        |
| 13.7.2 The FRVT database . . . . .  | 401        |
| 13.8 Experiments . . . . .  | 402        |
| 13.8.1 Data and Experimental Protocol . . . . .                             | 402        |
| 13.8.2 Euclidean Distance-Based Classifier . . . . .                        | 403        |
| 13.8.3 SVM-Based Classification . . . . .                                   | 405        |
| <br>  |            |
| <b>References . . . . .</b>   | <b>407</b> |
| <br>  |            |
| <b>14 Video Segmentation and Keyframe Extraction . . . . .</b>              | <b>413</b> |
| 14.1 Introduction . . . . .   | 413        |
| 14.2 Applications of Video Segmentation . . . . .                           | 414        |
| 14.3 Shot Boundary Detection . . . . .                                      | 416        |
| 14.3.1 Pixel-Based Approaches . . . . .                                     | 417        |
| 14.3.2 Block-Based Approaches . . . . .                                     | 418        |
| 14.3.3 Histogram-Based Approaches . . . . .                                 | 420        |
| 14.3.4 Clustering-Based Approaches . . . . .                                | 420        |
| 14.3.5 Performance Measures . . . . .                                       | 422        |
| 14.4 Shot Boundary Detection with <i>Torchvision</i> . . . . .              | 423        |
| 14.5 Keyframe Extraction . . . . .  | 424        |
| 14.6 Keyframe Extraction with <i>Torchvision</i> and <i>Torch</i> . . . . . | 426        |

References . . . . . 427

**Part IV Appendices**

**A Statistics** . . . . . 433

    A.1 Fundamentals . . . . . 433

        A.1.1 Probability and Relative Frequency . . . . . 433

        A.1.2 The Sample Space . . . . . 434

        A.1.3 The Addition Law . . . . . 434

        A.1.4 Conditional Probability . . . . . 437

        A.1.5 Statistical Independence . . . . . 438

    A.2 Random Variables . . . . . 439

        A.2.1 Fundamentals . . . . . 439

        A.2.2 Mathematical Expectation . . . . . 441

        A.2.3 Variance and Covariance . . . . . 442

**B Signal Processing** . . . . . 445

    B.1 Introduction . . . . . 445

    B.2 The Complex Numbers . . . . . 445

    B.3 The  $z$ -Transform . . . . . 447

        B.3.1  $z$ -Transform Properties . . . . . 449

        B.3.2 The Fourier Transform . . . . . 451

        B.3.3 The Discrete Fourier Transform . . . . . 452

    B.4 The Discrete Cosine Transform . . . . . 453

**C Matrix Algebra** . . . . . 457

    C.1 Introduction . . . . . 457

    C.2 Fundamentals . . . . . 457

    C.3 Determinants . . . . . 458

    C.4 Eigenvalues and Eigenvectors . . . . . 460

**D Mathematical Foundations of Kernel Methods** . . . . . 463

    D.1 Introduction . . . . . 463

    D.2 Scalar Products, Norms and Metrics . . . . . 464

    D.3 Positive Definite Kernels and Matrices . . . . . 465

        D.3.1 How to Make a Mercer Kernel . . . . . 469

    D.4 Conditionate Positive Definite Kernels and Matrices . . . . . 471

    D.5 Negative Definite Kernels and Matrices . . . . . 472

    D.6 Relations Between Positive and Negative  
        Definite Kernels . . . . . 474

    D.7 Metric Computation by Mercer Kernels . . . . . 476

    D.8 Hilbert Space Representation of Positive  
        Definite Kernels . . . . . 478

    D.9 Conclusions . . . . . 480

**References** ..... 481

**Index** ..... 483

## Introduction

### 1.1 Two Fundamental Questions

There are two fundamental questions that should be answered before buying, and even more before reading, a book:

- Why should one read the book?
- What is the book about?

This is the reason why this section, the first of the whole text, proposes some motivations for potential readers (Section 1.1.1) and an overall description of the content (Section 1.1.2). If the answers are convincing, further information can be found in the rest of this chapter: Section 1.2 shows in detail the structure of the book, Section 1.3 presents some features that can help the reader to better move through the text, and Section 1.4 provides some reading tracks targeting specific topics.

#### 1.1.1 Why Should One Read The Book?

One of the most interesting technological phenomena in recent years is the diffusion of consumer electronic products with constantly increasing acquisition, storage and processing power. As an example, consider the evolution of digital cameras: the first models available in the market in the early nineties produced images composed of 1.6 million pixels (this is the meaning of the expression *1.6 megapixels*), carried an onboard memory of 16 megabytes, and had an average cost higher than 10,000 U.S. dollars. At the time this book is being written, the best models are close to or even above 8 megapixels, have internal memories of one gigabyte and they cost around 1,000 U.S. dollars. In other words, while resolution and memory capacity have been multiplied by around five and fifty, respectively, the price has been divided by more than ten. Similar trends can be observed in all other kinds of digital devices including videocameras, cellular phones, mp3 players, personal digital assistants (PDA),

etc. *As a result, large amounts of digital material are being accumulated and need to be managed effectively in order to avoid the problem of information overload.*

The same period has witnessed the development of the Internet as ubiquitous source of information and services. In the early stages (beginning of the nineties), the webpages were made essentially of text. The reason was twofold: on the one hand the production of digital data different from simple texts was difficult (see above); on the other hand the connections were so slow that the download of a picture rather than an audio file was a painful process. Needless to say, how different the situation is today: multimedia material (including images, audio and videos) can be not only downloaded from the web from a computer, but also through cellular phones and PDAs. *As a consequence, the data must be adapted to new media with tight hardware and bandwidth constraints.*

The above phenomena have led to two major challenges for the scientific community:

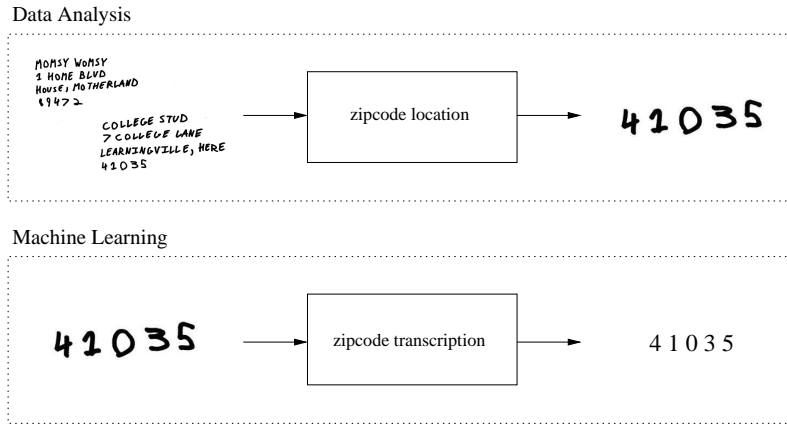
- *Data analysis*: it is not possible to take profit from large amounts of data without effective approaches for accessing their content. The goal of data analysis is to extract the data *content*, i.e. any information that constitutes an asset for potential users.
- *Data processing*: the data are an actual asset if they are accessible everywhere and available at any moment. This requires representing the data in a form that enables the transmission through physical networks as well as wireless channels.

This book addresses the above challenges, with a major emphasis on the analysis, and this is the main reason for reading this text. Moreover, even if the above challenges are among the hottest issues in current research, the techniques presented in this book enable one to address many other engineering problems involving complex data: automatic reading of handwritten addresses in postal plants, modeling of human actions in surveillance systems, analysis of historical documents archives, remote sensing (i.e. extraction of information from satellite images), etc. The book can thus be useful to almost any person dealing with audio, image and video data: students at the early stage of their education that need to lay the ground of their future career, PhD students and researchers who need a reference in their everyday activity, practitioners that want to keep the pace of the *state-of-the-art*.

### 1.1.2 What Is the Book About?

A first and general answer to the question '*What is the book about?*' can be obtained by defining the two parts of the title, i.e. *machine learning* (ML) on one side and *audio, image and video analysis* on the other side (for a more detailed description of the content of chapters see Section 1.2):





**Fig. 1.1.** Zipcode reading machine. The structure of the machine underlies the structure of the book: Part I involves the early stages of the data analysis block, Part II focuses on the machine learning block and Part III shows examples of other systems.

- ML is a multidisciplinary approach, involving several scientific domains (e.g. mathematics, computer science, physics, biology, etc.), that enable computers to automatically *learn from data*. By *learning* we mean here a process that takes as input data and gives as output algorithms capable of performing, over the same kind of data, a desired task.
- Image, audio and video analysis include any technique capable of extracting from the data *high-level* information, i.e. information that is not explicitly stated, but it requires an *abstraction process*.

As an example, consider a machine for the automatic transcription of zipcodes written on envelopes. Such machines route the letters towards their correct destination without human intervention and speed up significantly the mail delivery process.

The general scheme of such a machine is depicted in Figure 1.1 and it shows how both components of the title are involved: the image analysis part takes as input the digital image of the envelope and gives as output the regions actually containing the zipcode. From the point of view of the machine, the image is nothing other than an array of numbers and the position of the zipcode, then of its digits, is not explicitly available. The location of the zipcode is thus an operation that requires, following the above definition, an abstraction process.

The second stage is the actual transcription of the digits. Handwritten data are too variable and ambiguous to be transcribed with rules, i.e. with explicit conditions that must be met in order to transcribe a digit in one way rather than another. ML techniques address such a problem by using statistics to model large amounts of elementary information, e.g. the value of single pixels, and their relations.

The example concerns a problem where the data are images, but similar approaches can be found also for audio recordings and videos. In all cases, analysis and ML components interact in order to first convert the raw data into a format suitable for ML, and then apply ML techniques in order to perform a task of interest.

In summary, this book is about techniques that enable one to perform complex tasks over challenging data like audio recordings, images and videos data where the informations to be extracted are never explicit, but rather hidden behind the data statistical properties.

## 1.2 The Structure of the Book

The structure of the machine shown as an example in Section 1.1.2 underlies the structure of the book. The text is composed of three following parts:

- *From Perception to Computation.* This part shows how complex data such as audio, images and videos can be converted into mathematical objects suitable for computer processing and, in particular, for the application of ML techniques.
- *Machine Learning.* This part presents a wide selection of the machine learning approaches which are, in our opinion, most effective for image, video and audio analysis. Comprehensive surveys of ML are left to specific handbooks (see the references in Chapter 4).
- *Applications.* This part presents few major applications including ML and analysis techniques: handwriting and speech recognition, face recognition, video segmentation and keyframe extraction.

The book is then completed by four appendices that provide notions about the main mathematical instruments used throughout the text: signal processing, matrix algebra, probability theory and kernel theory. The following sections describe in more detail the content of each part.

### 1.2.1 Part I: From Perception to Computation

This part includes the following two chapters:

- Chapter 2: Audio Acquisition, Representation and Storage
- Chapter 3: Image and Video Acquisition, Representation and Storage

The main goal of this part is to show how the physical supports of our auditory and visual perceptions, i.e. acoustic waves and electromagnetic radiation, are converted into objects that can be manipulated by a computer. This is the sense of the name *From Perception to Computation*.

Chapter 2 focuses on audio data and starts with a description of the human auditory system. This shows how the techniques used to represent and store audio data try to capture the same information that seems to be most

important for human ears. Major attention is paid to the most common audio formats and their underlying encoding technologies. The chapter includes also some algorithms to perform basic operations such as silence detection in spoken data.

Chapter 3 focuses on images and videos and starts with a description of the human visual apparatus. The motivation is the same as in the case of audio data, i.e. to show how the way humans perceive images influences the engineering approaches to image acquisition, representation and storage. The rest of the chapter is dedicated to color models, i.e. the way visual sensations are represented in a computer, and to the most important image and video formats.

In terms of the machine depicted in Figure 1.1, Part I concerns the early steps of the analysis stage.

### 1.2.2 Part II: Machine Learning

This part includes the following chapters:

- Chapter 4: Machine Learning
- Chapter 5: Bayesian Decision Theory
- Chapter 6: Clustering Methods
- Chapter 7: Foundations of Statistical Machine Learning
- Chapter 8: Supervised Neural Networks and Ensemble Methods
- Chapter 9: Kernel Methods
- Chapter 10: Markovian Models for Sequential Data
- Chapter 11: Feature Extraction and Manifold Learning Methods

The main goal of Part II is to provide an extensive survey of the main techniques applied in machine learning. The chapters of Part II cover most of the ML algorithms applied in state-of-the-art systems for audio, image and video analysis.

Chapter 4 explains what machine learning is. It provides the basic terminology necessary to read the rest of the book, and introduces few fundamental concepts such as the difference between *supervised* and *unsupervised learning*.

Chapter 5 lays the groundwork on which most of the ML techniques are built, i.e. the *Bayesian decision theory*. This is a probabilistic framework where the problem of making decisions about the data, i.e. of deciding whether a given bitmap shows a handwritten "3" or another handwritten character, is stated in terms of probabilities.

Chapter 6 presents the so-called *clustering methods*, i.e. techniques that are capable of splitting large amounts of data, e.g. large collections of handwritten digit images, into groups called *clusters* supposed to contain only similar samples. In the case of handwritten digits, this means that all samples grouped in a given cluster should be of the same kind, i.e. they should all show the same digit.

Chapter 7 introduces two fundamental tools for assessing the performance of an ML algorithm: The first is the *bias-variance* decomposition and the second is the *Vapnik-Cervonenkis* dimension. Both instruments address the problem of *model selection*, i.e. finding the most appropriate model for the problem at hand.

Chapter 8 describes some of the most popular ML algorithms, namely *neural networks* and *ensemble techniques*. The first is a corpus of techniques inspired by the organization of the neurons in the brain. The second is the use of multiple algorithms to achieve a collective performance higher than the performance of any single item in the ensemble.

Chapter 9 introduces the *kernel methods*, i.e. techniques based on the projection of the data into spaces where the tasks of interest can be performed better than in the original space where they are represented.

Chapter 10 shows a particular class of ML techniques, the so-called *Markovian models*, which aim at modeling sequences rather than single objects. This makes them particularly suitable for any problem where there are temporal or spatial constraints.

Chapter 11 presents some techniques that are capable of representing the data in a form where the actual information is enhanced while the noise is eliminated or at least attenuated. In particular, these techniques aim at reducing the data dimensionality, i.e. the number of components necessary to represent the data as vectors. This has several positive consequences that are described throughout the chapter.

In terms of the machine depicted in Figure 1.1, Part II addresses the problem of transcribing the zipcode once it has been located by the analysis part.

### 1.2.3 Part III: Applications

Part II includes the following chapters:

- Chapter 12: Speech and Handwriting Recognition
- Chapter 13: Face Recognition
- Chapter 14: Video Segmentation and Keyframe Extraction

The goal of Part III is to present examples of applications using the techniques presented in Part II. Each chapter of Part III shows an overall system where analysis and ML components interact in order to accomplish a given task. Whenever possible, the chapters of this part present results obtained using publicly available data and software packages. This enables the reader to perform experiments similar to those presented in this book.

Chapter 12 shows how Markovian models are applied to the automatic transcription of spoken and handwritten data. The goal is not only to present two of the most investigated problems of the literature, but also to show how the same technique can be applied to two kinds of data apparently different like speech and handwriting.

Chapter 13 presents *face recognition*, i.e. the problem of recognizing the identity of a person portrayed in a digital picture. The algorithms used in this chapter are the principal component analysis (one of the feature extraction methods shown in Chapter 11) and the support vector machines (one of the algorithms presented in Chapter 9).

Chapter 14 shows how clustering techniques are used for the segmentation of videos into shots<sup>1</sup> and how the same techniques are used to extract from each shot the most representative image.

Each chapter presents an application as a whole, including both analysis and ML components. In other words, Part III addresses elements that can be found in all stages of Figure 1.1.

#### 1.2.4 Appendices

The four appendices at the end of the book provide the main notions about the mathematical instruments used throughout the book:

- *Appendix A: Signal Processing.* This appendix presents the main elements of signal processing theory including Fourier transform, z-transform, discrete cosine transform and a quick recall of the complex numbers. This appendix is especially useful for reading Chapter 2 and Chapter 12.
- *Appendix B: Statistics.* This appendix introduces the main statistical notions including space of the events, probability, mean, variance, statistical independence, etc. The appendix is useful to read all chapters of Parts II and III.
- *Appendix C: Matrix Algebra.* This appendix gives basic notions on matrix algebra and provides a necessary support for going through some of the mathematical procedures shown in Part II.
- *Appendix D: Kernel Theory.* This appendix presents kernel theory and it is the natural complement of Chapter 9.

None of the appendices present a complete and exhaustive overview of the domain they are dedicated to, but they provide sufficient knowledge to read all the chapters of the book. In other words, the goal of the appendices is not to replace specialized monographies, but to make this book as self-consistent as possible.

### 1.3 How to Read This Book

This section explains some features of this book that should help the reader to better move through the different parts of the text:

---

<sup>1</sup> A shot is an unbroken sequence of images captured with a video camera.

- *Background and Learning Goal Information*: at the beginning of each chapter, the reader can find information about required background and learning goals.
- *Difficulty Level of Each Section*: sections requiring a deeper mathematical background are signaled.
- *Problems*: at the end of the chapters of Parts I and II (see Section 1.2) there are problems aimed at testing the skills acquired by reading the chapter.
- *Software*: whenever possible, the text provides pointers to publicly available data and software packages. This enable the reader to immediately put in practice the notions acquired in the book.

The following sections provide more details about each of the above features.

### 1.3.1 Background and Learning Objectives

At the beginning of each chapter, the reader can find two lists: the first is under the header *What the reader should know **before** reading this chapter*, the second is under the header *What the reader should know **after** reading this chapter*. The first list provides information about the preliminary notions necessary to read the chapter. The book is mostly self-contained and the background can often be found in other chapters or in the appendices. However, in some cases the reader is expected to have the basic knowledge provided in the average undergraduate studies. The second list sets a certain number of goals to be achieved by reading the chapter. The objectives are designed to be a measure of a correct understanding of the chapter content.

### 1.3.2 Difficulty Level

The titles of some sections show a \* or \*\* symbol at the end.<sup>2</sup> The meaning is that the content of the sections requires a background available only at the end of the undergraduate studies (one star) or at the level of PhD and beyond (two stars). This is not supposed to discourage the readers, but rather to help them to better focus on the sections that are more accessible to them. On the other hand, the assignment of the difficulty level is mostly based on the experience of the authors. Graduate and undergraduate study programs are different depending on universities and countries and what the authors consider *difficult* can be considered *accessible* in other situations. In other words, the difficulty level has to be considered a warning rather than a prescription.

### 1.3.3 Problems

At the end of each chapter, the reader can find some problems. In some cases the problems propose to demonstrate theorems or to solve exercises, in other

---

<sup>2</sup> Sections with no stars are supposed to be accessible to anybody.

cases they propose to perform experiments using publicly available software packages (see below).

### 1.3.4 Software

Whenever possible, the book provides pointers to publicly available software packages and data. This should enable the readers to immediately apply in practice the algorithms and the techniques shown in the text. All packages are widely used in the scientific community and are accompanied by extensive documentation (provided by the package authors). Moreover, since data and packages have typically been applied in several works presented in the literature, the readers have the possibility to repeat the experiments performed by other researchers and practitioners.

## 1.4 Reading Tracks

The book is not supposed to be read as a whole. Readers should start from their needs and identify the chapters most likely to address them. This section provides few reading tracks targeted at developing specific competences. Needless to say, the tracks are simply suggestions and provide an orientation through the content of the book, rather than a rigid prescription.

- *Introduction to Machine Learning*. This track includes Appendix A, and Chapters 4, 5 and 7:
  - *Target Readers*: students and practitioners that study machine learning for the first time.
  - *Goal*: to provide the first and fundamental notions about ML, including what ML is, what can be done with ML, and what are the problems that can be addressed using ML.
- *Kernel Methods and Support Vector Machines*. This track includes Appendix D, Chapter 7 and Chapter 9. Chapter 13 is optional.
  - *Target Readers*: experienced ML practitioners and researchers that want to include kernel methods in their toolbox or background.
  - *Goal*: to provide competences necessary to understand and use support vector machines and kernel methods. Chapter 13 provides an example of application, i.e. automatic face recognition, and pointers to free packages implementing support vector machines.
- *Markov Models for Sequences*. This track includes Appendix A, Chapter 5 and Chapter 10. Chapter 12 is optional.
  - *Target Readers*: experienced ML practitioners and researchers that want to include Markov models in their toolbox or background.
  - *Goal*: to provide competences necessary to understand and use hidden Markov models and  $N$ -gram models. Chapter 12 provides an example of application, i.e. handwriting recognition, and describes free packages implementing Markov models.

- *Unsupervised Learning Techniques.* This track includes Appendix A, Chapter 5 and Chapter 6. Chapter 14 is optional.
  - *Target Readers:* experienced ML practitioners and researchers that want to include clustering techniques in their toolbox or background.
  - *Goal:* to provide competences necessary to understand and use the main unsupervised learning techniques. Chapter 14 provides an example of application, i.e. shot detection in videos.
- *Data processing.* This track includes Appendix B, Chapter 2 and 3.
  - *Target Readers:* students, researchers and practitioners that work for the first time with audio and images.
  - *Goal:* to provide the basic competences necessary to acquire, represent and store audio files and images.

## Acknowledgements

This book would not have been possible without the help of several persons. First of all we wish to thank Lakhmi Jain and Catherine Brett who managed the book proposal submission. Then we thank those who helped us to significantly improve the original manuscript: the copyeditor at Springer-Verlag and our colleagues and friends Fabien Cardinaux, Matthias Dolder, Sarah Favre, Maurizio Filippone, Giulio Giunta, Itshak Lapidot, Guillaume Lathoud, Sébastien Marcel, Daniele Mastrangelo, Franco Masulli, Alexei Podzhnoukov, Guillermo Aradilla Zapata. Finally, we thank the Department of Applied Sciences at Parthenope University (Naples, Italy) and the IDIAP Research Institute (Martigny, Switzerland) for letting us dedicate a significant amount of time and energy to this book.



From Perception to Computation



## Audio Acquisition, Representation and Storage

### What the reader should know to understand this chapter

- Basic notions of physics.
- Basic notions of calculus (trigonometry, logarithms, exponentials, etc.)

### What the reader should know after reading this chapter

- Human hearing and speaking physiology.
- Signal processing fundamentals.
- Representation techniques behind the main audio formats.
- Perceptual coding fundamentals.
- Audio sampling fundamentals.

## 2.1 Introduction

The goal of this chapter is to provide basic notions about *digital audio processing technologies*. These are applied in many everyday life products such as phones, radio and television, videogames, CD players, cellular phones, etc. However, although there is a wide spectrum of applications, the main problems to be addressed in order to manipulate digital sound are essentially three: *acquisition*, *representation* and *storage*. The acquisition is the process of converting the physical phenomenon we call sound into a form suitable for digital processing, the representation is the problem of extracting from the sound information necessary to perform a specific task, and the storage is the problem of reducing the number of bits necessary to encode the acoustic signals.

The chapter starts with a description of the sound as a physical phenomenon (Section 2.2). This shows that acoustic waves are completely determined by the energy distribution across different frequencies; thus, any sound processing approach must deal with such quantities. This is confirmed by an

analysis of voicing and hearing mechanisms in humans. In fact, the vocal apparatus determines frequency and energy content of the voice through the *vocal folds* and the *articulators*. Such organs are capable of changing the shape of the vocal tract like it happens in the cavity of a flute when the player acts on keys or holes. In the case of sound perception, the main task of the ears is to detect the frequencies present in an incoming sound and to transmit the corresponding information to the brain. Both production and perception mechanisms have an influence on audio processing algorithms.

The acquisition problem is presented in Section 2.3 through the description of the *analog-to-digital* (A/D) conversion, the process transforming any analog signal into a form suitable for computer processing. Such a process is performed by measuring at discrete time steps the physical effects of a signal. In the case of the sound, the effect is the displacement of an elastic membrane in a microphone due to the pressure variations determined by acoustic waves. Section 2.3 presents the two main issues involved in the acquisition process: the first is the *sampling*, i.e. the fact that the original signal is continuous in time, but the effect measurements are performed only at discrete-time steps. The second is the *quantization*, i.e. the fact that the physical measurements are continuous, but they must be quantized because only a finite number of bits is available on a computer.

The quantization plays an important role also in storage problems because the number of bits used to represent a signal affects the amount of memory space needed to store a recording. Section 2.4 presents the main techniques used to store audio signals by describing the most common audio *formats* (e.g. *WAV*, *MPEG*, *mp3*, etc.). The reason is that each format corresponds to a different *encoding* technique, i.e. to a different way of representing an audio signal. The goal of encoding approaches is to reduce the amount of bits necessary to represent a signal while keeping an acceptable perceptual quality. Section 2.4 shows that the pressure towards the reduction of the *bit-rate* (the amount of bits necessary to represent one second of sound) is due not only to the emergence of new applications characterized by tighter space and bandwidth constraints, but also by consumer preferences.

While acquisition and storage problems are solved with relatively few standard approaches, the representation issue is task dependent. For storage problems (see above), the goal of the representation is to preserve as much as possible the information of the acoustic waveforms, in prosody analysis or topic segmentation, it is necessary to detect the silences or the energy of the signal, in speaker recognition the main information is in the frequency content of the voice, and the list could continue. Section 2.5 presents some of the most important techniques analyzing the variations of the signal to extract useful information. The corpus of such techniques is called *time domain processing* in opposition to *frequency-domain* methods that work on the spectral representation of the signals and are shown in Appendix B and Chapter 12.

Most of the content of this chapter requires basic mathematical notions, but few points need familiarity with Fourier analysis. When this is the case,

the text includes a warning and the parts that can be difficult for unexperienced readers can be skipped without any problem. An introduction to Fourier analysis and frequency domain techniques is available in Appendix B. Each section provides references to specialized books and tutorials presenting in more detail the different issues.

## 2.2 Sound Physics, Production and Perception

This section presents the sound from both a physical and physiological point of view. The description of the main acoustic waves properties shows that the sound can be fully described in terms of frequencies and related energies. This result is obtained by describing the propagation of a single frequency sine wave, an example unrealistically simple, but still representative of what happens in more realistic conditions. In the following, this section provides a general description of how the human beings interact with the sound. The description concerns the way the speech production mechanism determines the frequency content of the voice and the way our ears detect frequencies in incoming sounds.

For more detailed descriptions of the acoustic properties, the reader can refer to more extensive monographies [3][16][24] and tutorials [2][11]. The psychophysiology of hearing is presented in [23][30], while good introductions to speech production mechanisms are provided in [9][17].

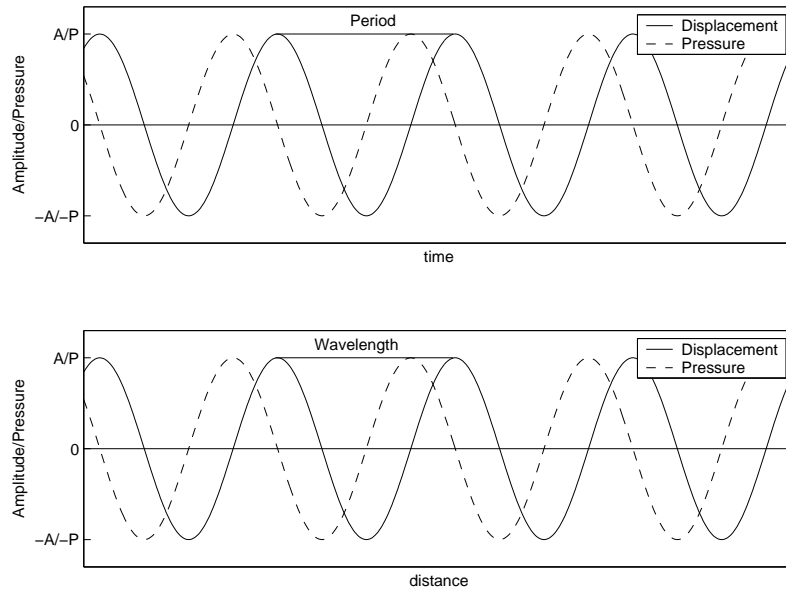
### 2.2.1 Acoustic Waves Physics

The physical phenomenon we call sound is originated by air molecule oscillations due to the mechanical energy emitted by an acoustic source. The displacement  $s(t)$  with respect to the equilibrium position of each molecule can be modeled as a sinusoid:

$$s(t) = A \sin(2\pi ft + \phi) = A \sin\left(\frac{2\pi}{T}t + \phi\right) \quad (2.1)$$

where  $A$  is called amplitude and represents the maximum distance from the equilibrium position (typically measured in *nanometers*),  $\phi$  is the phase,  $T$  is called period and it is the time interval length between two instants where  $s(t)$  takes the same value, and  $f = 1/T$  is the frequency measured in *Hz*, i.e. the number of times  $s(t)$  completes a cycle per second. The function  $s(t)$  is shown in the upper plot of Figure 2.1. Since all air molecules in a certain region of the space oscillate together, the acoustic waves determine local variations of the density that correspond to periodic compressions and rarefactions. The result is that the pressure changes with the time following a sinusoid  $p(t)$  with the same frequency as  $s(t)$ , but amplitude  $P$  and phase  $\phi^* = \phi + \pi/2$ :

$$p(t) = P \sin\left(2\pi ft + \phi + \frac{\pi}{2}\right) = P \sin\left(\frac{2\pi}{T}t + \phi + \frac{\pi}{2}\right). \quad (2.2)$$



**Fig. 2.1.** Frequency and wavelength. The upper plot shows the displacement of air molecules with respect to their equilibrium position as a function of time. The lower plot shows the distribution of pressure values as a function of the distance from the sound source.

The dashed sinusoid in the upper plot of Figure 2.1 corresponds to  $p(t)$  and it shows that the pressure variations have a delay of a quarter of period (due to the  $\pi/2$  added to the phase) with respect to  $s(t)$ . The maximum pressure variations correspond, for the highest energy sounds in a common urban environment, to around 0.6 percent of the atmospheric pressure.

When the air molecules oscillate, they transfer part of their mechanical energy to surrounding particles through collisions. The molecules that receive energy start oscillating and, with the same mechanism, they transfer mechanic energy to further particles. In this way, the acoustic waves propagate through the air (or any other medium) and can reach listeners far away from the source. The important aspect of such a propagation mechanism is that there is no net flow of particles no matter is transported from the point where the sound is emitted to the point where a listener receives it. Sound propagation is actually due to energy transport that determines pressure variations and molecule oscillations at distance  $x$  from the source.

The lower plot of Figure 2.1 shows the displacement  $s(x)$  of air molecules as a function of the distance  $x$  from the audio source:

$$s(x) = A \sin \left( \frac{2\pi}{v} f x + \phi \right) = A \sin \left( \frac{2\pi}{\lambda} x + \phi \right) \quad (2.3)$$

where  $v$  is the sound speed in the medium and  $\lambda = v/f$  is the *wavelength*, i.e. the distance between two points where  $s(x)$  takes the same value (the meaning of the other symbols is the same as in Equation (2.1)). Each point along the horizontal axis of the lower plot in Figure 2.1 corresponds to a different molecule of which  $s(x)$  gives the displacement. The pressure variation  $p(x)$  follows the same sinusoidal function, but has a quarter of period delay like in the case of  $p(t)$  (dashed curve in the lower plot of Figure 2.1):

$$p(x) = P \sin\left(\frac{2\pi}{v}fx + \phi + \frac{\pi}{2}\right) = P \sin\left(\frac{2\pi}{\lambda}x + \phi + \frac{\pi}{2}\right). \quad (2.4)$$

The equations of this section assume that an acoustic wave is completely characterized by two parameters: the frequency  $f$  and the amplitude  $A$ . From a perceptual point of view,  $A$  is related to the *loudness* and  $f$  corresponds to the *pitch*. While two sounds with equal loudness can be distinguished based on their frequency, for a given frequency, two sounds with different amplitude are perceived as the same sound with different loudness. The value of  $f$  is measured in *Hertz* (Hz), i.e. the number of cycles per second. The measurement of  $A$  is performed through the physical effects that depend on the amplitude like pressure variations.

The amplitude is related to the energy of the acoustic source. In fact, the higher is the energy, the higher is the displacement and, correspondently, the perceived loudness of the sound. From an audio processing point of view, the important aspect is what happens for a listener at a distance  $R$  from the acoustic source. In order to find a relationship between the source energy and the distance  $R$ , it is possible to use the *intensity*  $I$ , i.e. the energy passing per time unit through a surface unit. If the medium around the acoustic source is *isotropic*, i.e. it has the same properties along all directions, the energy is distributed uniformly on spherical surfaces of radius  $R$  centered in the source. The intensity  $I$  can thus be expressed as follows:

$$I(R) = \frac{W}{4\pi R^2} \quad (2.5)$$

where  $W = \Delta E/\Delta t$  is the source power, i.e. the amount of energy  $\Delta E$  emitted in a time interval of duration  $\Delta t$ . The power is measured in watts ( $W$ ) and the intensity in watts per square meter ( $W/m^2$ ). The relationship between  $I$  and  $A$  is as follows:

$$I = 2Z\pi^2 f^2 A^2 \quad (2.6)$$

where  $Z$  is a characteristic of the medium called *acoustic impedance*.

Since the only sounds that are interesting in audio applications are those that can be perceived by human beings, the intensities can be measured through their ratio  $I/I_0$  to the *threshold of hearing* (THO)  $I_0$ , i.e. the minimum intensity detectable by human ears. However, this creates a problem because the value of  $I_0$  corresponds to  $10^{-12} W/m^2$ , while the maximum value of  $I$  that can be tolerated without permanent physiological damages is

$I_{max} = 10^3 \text{ W/m}^2$ . The ratio  $I/I_0$  can thus range across 15 orders of magnitude and this makes it difficult to manage different intensity values. For this reason, the ratio  $I/I_0$  is measured using the *deciBel* (*dB*) scale:

$$I^* = 10 \log_{10} \left( \frac{I}{I_0} \right) \quad (2.7)$$

where  $I^*$  is the intensity measured in *dB*. In this way, the intensity values range between 0 ( $I = I_0$ ) and 150 ( $I = I_{max}$ ). Since the intensity is proportional to the square power of the maximum pressure variation  $P$  as follows:

$$I = \frac{P^2}{2Z}, \quad (2.8)$$

the value of  $I^*$  can be expressed also in terms of *db SPL* (sound pressure level):

$$I^* = 20 \log_{10} \left( \frac{P}{P_0} \right). \quad (2.9)$$

The numerical value of the intensity is the same when using *dB* or *db SPL*, but the latter unit allows one to link intensity and pressure. This is important because the pressure is a physical effect relatively easy to measure and the microphones rely on it (see Section 2.3).

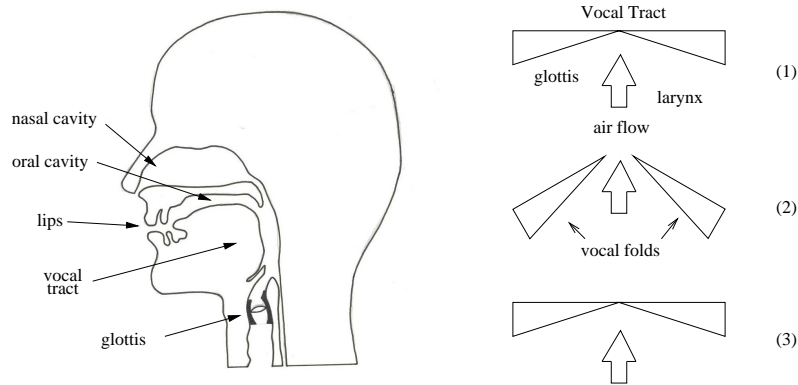
Real sounds are never characterized by a single frequency  $f$ , but by an energy distribution across different frequencies. In intuitive terms, a sound can be thought of as a “sum of single frequency sounds,” each characterized by a specific frequency and a specific energy (this aspect is developed rigorously in Appendix B). The important point of this section is that a sound can be fully characterized through frequency and energy measures and the next sections show how the human body interacts with sound using such informations.

### 2.2.2 Speech Production

Human voices are characterized, like any other acoustic signal, by the energy distribution across different frequencies. This section provides a high-level sketch of how the human vocal apparatus determines such characteristics. Deeper descriptions, especially from the anatomy point of view, can be found in specialized monographies [23][30].

The voice mechanism starts when the diaphragm pushes air from lungs towards the oral and nasal cavities. The air flow has to pass through an organ called *glottis* that can be considered like a gate to the *vocal tract* (see Figure 2.2). The glottis determines the frequency distribution of the voice, while the vocal tract (composed of larynx and oral cavity) is at the origin of the energy distribution across frequencies. The main components of the glottis are the vocal folds and the way they react with respect to air coming from the lungs enables to distinguish between the two main classes of sounds produced by human beings. When the vocal folds vibrate, the sounds are called *voiced*,





**Fig. 2.2.** Speech production. The left figure shows a sketch of the speech production apparatus (picture by Matthias Dolder); the right figure shows the glottal cycle: the air flow increases the pressure below the glottis (1), the vocal folds open to reequilibrate the pressure difference between larynx and vocal tract (2), once the equilibrium is achieved the vocal folds close again (3). the cycle is repeated as long as air is pushed by the lungs.

otherwise they are called *unvoiced*. For a given language, all words can be considered like sequences of elementary sounds, called *phonemes*, belonging to a finite set that contains, for western languages, 35-40 elements on average and each phoneme is either voiced or unvoiced.

When a voiced phoneme is produced, the vocal folds vibrate following the cycle depicted in Figure 2.2. When air arrives at the glottis, the pressure difference with respect to the vocal tract increases until the vocal folds are forced to open to reestablish the equilibrium. When this is reached, the vocal folds close again and the cycle is repeated as long as voiced phonemes are produced. The vibration frequency of the vocal folds is a characteristic specific of each individual and it is called *fundamental frequency*  $F_0$ , the single factor that contributes more than anything else to the voice pitch. Moreover, most of the energy in human voices is distributed over the so-called *formants*, i.e. sound components with frequencies that are integer multiples of  $F_0$  and correspond to the resonances of the vocal tract. Typical  $F_0$  values range between 60 and 300 Hz for adult men and small children (or adult women) respectively. This means that the first 10-12 formants, on which most of the speech energy is distributed, correspond to less than 4000 Hz. This has important consequences on the human auditory system (see Section 2.2.3) as well as on the design of speech acquisition systems (see Section 2.3).

The production of unvoiced phonemes does not involve the vibration of the vocal folds. The consequence is that the frequency content of unvoiced phonemes is not as defined and stable as the one of voiced phonemes and that their energy is, on average, lower than that of the others. Examples of voiced phonemes are the vowels and the phonemes corresponding to the first

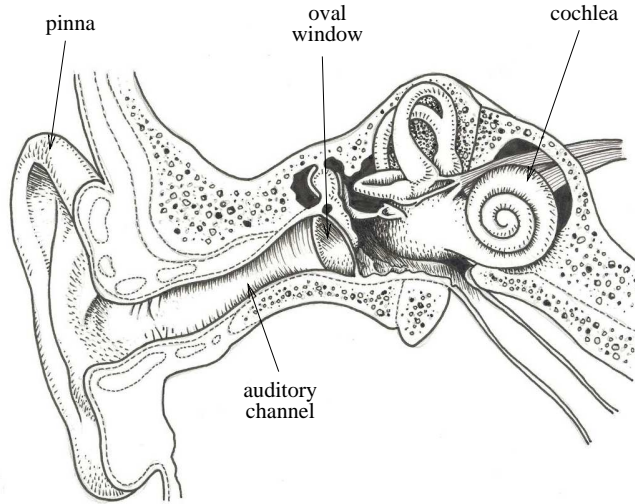
sound in words like *milk* or *lag*, while unvoiced phonemes can be found at the beginning of words *six* and *stop*. As a further example you can consider the words *son* and *zone* which have phonemes at the beginning where the vocal tract has the same configuration, but in the first case (*son*) the initial phoneme is unvoiced, while it is voiced in the second case. The presence of unvoiced phonemes at the beginning or the end of words can make it difficult to detect their boundaries.

The sounds produced at the glottis level must still pass through the vocal tract where several organs play as *articulators* (e.g. tongue, lips, velum, etc.). The position of such organs is defined *articulators configuration* and it changes the shape of the vocal tract. Depending on the shape, the energy is concentrated on certain frequencies rather than on others. This makes it possible to reconstruct the articulator configuration at a certain moment by detecting the frequencies with the highest energy. Since each phoneme is related to a specific articulator configuration, energy peak tracking, i.e. the detection of highest energy frequencies along a speech recording, enables, in principle, to reconstruct the voiced phoneme sequences and, since most speech phonemes are voiced, the corresponding words. This will be analyzed in more detail in Chapter 12.

### 2.2.3 Sound Perception

This section shows how the human auditory peripheral system (APS), i.e. what the common language defines as *ears*, detects the frequencies present in incoming sounds and how it reacts to their energies (see Figure 2.3). The definition *peripheral* comes from the fact that no cognitive functions, performed in the brain, are carried out at its level and its only role is to acquire the information contained in the sounds and to transmit it to the brain. In machine learning terms, the ear is a basic *feature extractor* for the brain. The description provided here is just a sketch and more detailed introductions to the topic can be found in other texts [23][30].

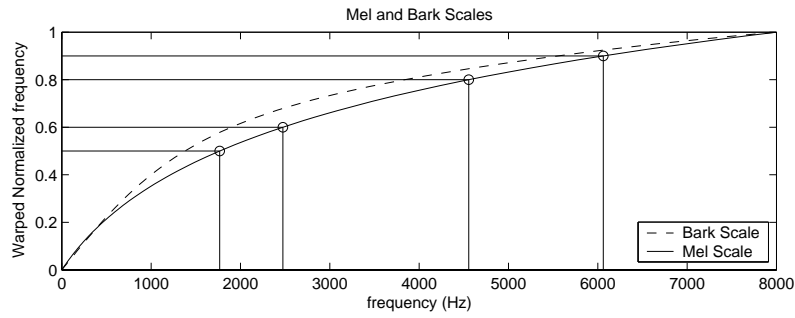
The APS is composed of three parts called *outer*, *middle* and *inner* ear. The outer ear is the *pinna* that can be observed at both sides of the head. Following recent experiments, the role of the outer ear, considered minor so far, seems to be important in the detection of the sound sources position. The middle ear consists of the auditory channel, roughly 1.3 cm long, which connects the external environment with the inner ear. Although it has such a simple structure, the middle ear has two important properties, the first is that it optimizes the transmission of frequencies between around 500 and 4000 Hz, the second is that it works as an impedance matching mechanism with respect to the inner ear. The first property is important because it makes the APS particularly effective in hearing human voices (see previous section), the second one is important because the inner ear has an acoustic impedance higher than air and all the sounds would be reflected at its entrance without an impedance matching mechanism.



**Fig. 2.3.** Auditory peripheral system. The peripheral system can be divided into outer (the pinna is the ear part that can be seen on the sides of the head), middle (the channel bringing sounds toward the cochlea) and inner part (the cochlea and the hair cells). Picture by Matthias Dolder.

The main organ of the inner ear is the *cochlea*, a bony spiral tube around 3.5 cm long that coils 2.6 times. Incoming sounds penetrate into the cochlea through the *oval window* and propagate along the *basilar membrane* (BM), an elastic membrane that follows the spiral tube from the *base* (in correspondence of the oval window) to the *apex* (at the opposite extreme of the tube). In the presence of incoming sounds, the BM vibrates with an amplitude that changes along the tube. At the base the amplitude is at its minimum and it increases constantly until a maximum is reached, after which point the amplitude decreases quickly so that no more vibrations are observed in the rest of the BM length. The important aspect of such a phenomenon is that the point where the maximum BM displacement is observed depends on the frequency. In other words, the cochlea operates a *frequency-to-place* conversion that associates each frequency  $f$  to a specific point of the BM. The frequency that determines a maximum displacement at a certain position is called the *characteristic frequency* for that place. The nerves connected to the external cochlea walls in correspondence of such a point are excited and the information about the presence of  $f$  is transmitted to the brain.

The frequency-to-place conversion is modeled in some popular speech processing algorithms through the *critical band analysis*. In such an approach, the cochlea is modeled as a bank of bandpass filters, i.e. as a device composed of several filters stopping all frequencies outside a predefined interval called



**Fig. 2.4.** Frequency normalization. Uniform sampling on the vertical axis induces on the horizontal axis frequency intervals more plausible from a perceptual point of view. Frequencies are sampled more densely when they are lower than  $4kHz$ , the region covered by the human auditory system.

*critical band* and centered around a *critical frequency*  $f_j$ . The problem of finding appropriate  $f_j$  values is addressed by selecting frequencies such that the perceptual difference between  $f_i$  and  $f_{i+1}$  is the same for all  $i$ . This condition can be achieved by mapping  $f$  onto an appropriate scale  $T(f)$  and by selecting frequency values such that  $T(f_{i+1}) - T(f_i)$  has the same values for every  $i$ . The most popular transforms are the *Bark scale*:

$$b(f) = 13 \cdot \arctan(0.00076f) + 3.5 \cdot \arctan\left(\frac{f^2}{7500^2}\right), \quad (2.10)$$

and the *Mel scale*

$$B(f) = 1125 \cdot \ln\left(1 + \frac{f}{700}\right). \quad (2.11)$$

Both above functions are plotted in Figure 2.4 and have finer resolution at lower frequencies. This means that ears are more sensitive to differences at low frequencies than at high frequencies.

### 2.3 Audio Acquisition

This section describes the audio *acquisition* process, i.e. the conversion of sound waves, presented in the previous section from a physical and physiological point of view, into a format suitable for machine processing. When the machine is a digital device, e.g. computers and *digital signal processors* (DSP), such a process is referred to as *analog-to-digital* (A/D) conversion because an analogic signal (see below for more details) is transformed into a digital object, e.g. a series of numbers. In general, the A/D conversion is performed by measuring one or more physical effects of a signal at discrete time steps. In the case of the acoustic waves, the physical effect that can be measured

more easily is the pressure  $p$  in a certain point of the space. Section 2.2 shows that the signal  $p(t)$  has the same frequency as the acoustic wave at its origin. Moreover, it shows that the square of the pressure is proportional to the sound intensity  $I$ . In other words, the pressure variations capture the information necessary to fully characterize incoming sounds.

In order to do this, microphones contain an elastic membrane that vibrates when the pressure at its sides is different (this is similar to what happens in the ears where an organ called *eardrum* captures pressure variations). The displacement  $s(t)$  at time  $t$  of a membrane point with respect to the equilibrium position is proportional to the pressure variations due to incoming sounds, thus it can be used as an indirect measure of  $p$  at the same instant  $t$ . The result is a signal  $s(t)$  which is continuous in time and takes values over a continuous interval  $S = [-S_{max}, S_{max}]$ . On the other hand, the measurement of  $s(t)$  can be performed only at specific instants  $t_i$  ( $i = 0, 1, 2, \dots, N$ ) and no information is available about what happens between  $t_i$  and  $t_{i+1}$ . Moreover, the displacement measures can be represented only with a finite number  $B$  of bits, thus only  $2^B$  numbers are available to represent the non countable values of  $S$ . The above problems are called *sampling* and *quantization*, respectively, and have an important influence on the acquisition process. They can be studied separately and are introduced in the following sections.

Extensive descriptions of the acquisition problem can be found in signal processing [22][28] and speech recognition [15] books.

### 2.3.1 Sampling and Aliasing

During the sampling process, the displacement of the membrane is measured at regular time steps. The number  $F$  of measurements per second is called *sampling frequency* or *sampling rate* and, correspondently, the length  $T_c = 1/F$  of the time interval between two consecutive measurements is called *sampling period*. The relationship between the analog signal  $s(t)$  and the sampled signal  $s[n]$  is as follows:

$$s[n] = s(nT_c) \quad (2.12)$$

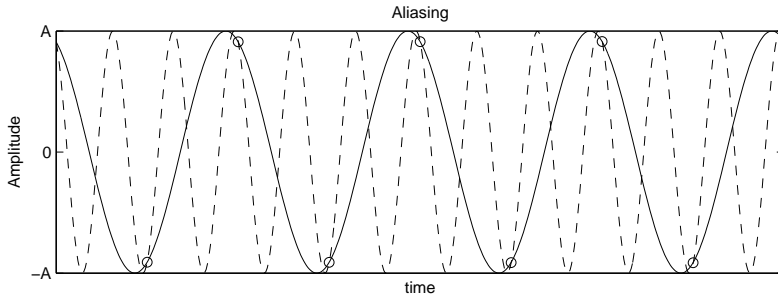
where the square brackets are used for sampled discrete-time signals and the parentheses are used for continuous signals (the same notation will be used throughout the rest of this chapter).

As an example, consider a sinusoid  $s(t) = A \sin(2\pi ft + \phi)$ . After the sampling process, the resulting digital signal is:

$$s[n] = A \sin(2\pi fnT_c + \phi) = A \sin(2\pi f_0 n + \phi) \quad (2.13)$$

where  $f_0 = f/F$  is called *normalized frequency* and it corresponds to the number of sinusoid cycles per sampling period. Consider now the infinite set of continuous signals defined as follows:

$$s_k(t) = A \sin(2k\pi Ft + 2\pi ft + \phi) \quad (2.14)$$



**Fig. 2.5.** Aliasing. Two sinusoidal signals are sampled at the same rate  $F$  and result in the same sequence of points (represented with circles).

where  $k \in (0, 1, \dots, \infty)$ , and the corresponding digital signals sampled at frequency  $F$ :

$$s_k[n] = A \sin(2k\pi n + 2\pi f_0 n + \phi). \quad (2.15)$$

Since  $\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$ , the sinus of a multiple of  $2\pi$  is always null, and the cosine of a multiple of  $2\pi$  is always 1, the last equation can be rewritten as follows:

$$s_k[n] = A \sin(2\pi f_0 n + \phi) = s[n] \quad (2.16)$$

where  $k \in (0, 1, \dots, \infty)$ , then there are infinite sinusoidal functions that are transformed into the same digital signal  $s[n]$  through an A/D conversion performed at the same rate  $F$ .

Such problem is called *aliasing* and it is depicted in Figure 2.5 where two sinusoids are shown to pass through the same points at time instants  $t_n = nT$ . Since every signal emitted from a natural source can be represented as a sum of sinusoids, the aliasing can possibly affect the sampling of any signal  $s(t)$ . This is a major problem because does not allow a one-to-one mapping between incoming and sampled signals. In other words, different sounds recorded with a microphone can result, once they have been acquired and stored on a computer, into the same digital signal.

However, the problem can be solved by imposing a simple constraint on  $F$ . Any acoustic signal  $s(t)$  can be represented as a superposition of sinusoidal waves with different frequencies. If  $f_{max}$  is the highest frequency represented in  $s(t)$ , the aliasing can be avoided if:

$$F > 2f_{max} \quad (2.17)$$

where  $2f_{max}$  is called the *critical frequency*, *Nyquist frequency* or *Shannon frequency*. The inequality is strict; thus the aliasing can still affect the sampling process when  $F = 2f_{max}$ . In practice, it is difficult to know the value of  $f_{max}$ , then the microphones apply a low-pass filter that eliminates all frequencies

below a certain threshold that corresponds to less than  $F/2$ . In this way the condition in Equation (2.17) is met.<sup>1</sup>

The demonstration of the fact that the condition in Equation (2.17) enables us to avoid the aliasing problem is given in the so-called *sampling theorem*, one of the foundations of signal processing. Its demonstration is given in the next subsection and it requires some deeper mathematical background. However, it is not necessary to know the demonstration to understand the rest of this chapter; thus unexperienced readers can go directly to Section 2.3.3 and continue the reading without problems.

### 2.3.2 The Sampling Theorem\*\*

Aliasing is due to the effect of sampling in the frequency domain. In order to identify the conditions that enable to establish a one-to-one relationship between continuous signals  $s(t)$  and corresponding digital sampled sequences  $s[n]$ , it is thus necessary to investigate the relationship between the Fourier transforms of  $s(t)$  and  $s[n]$  (see Appendix B).

The FT of  $s(t)$  is given by:

$$S_a(j\omega) = \int_{-\infty}^{\infty} s(t)e^{-j\omega t} dt, \quad (2.18)$$

while the FT of the sampled signal is:

$$S_d(e^{j\omega}) = \sum_{n=-\infty}^{\infty} s[n]e^{-j\omega n}. \quad (2.19)$$

However, the above  $S_d$  form is not the most suitable to show the relationship with  $S_a$ , thus we need to find another expression. The sampling operation can be thought of as the product between the continuous signal  $s(t)$  and a *periodic impulse train* (PIT)  $p(t)$ :

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_c), \quad (2.20)$$

where  $T_c$  is the sampling period, and  $\delta(k) = 1$  for  $k = 0$  and  $\delta(k) = 0$  otherwise. The result is a signal  $s_p(t)$  that can be written as follows:

$$s_p(t) = s(t)p(t) = s(t) \sum_{n=-\infty}^{\infty} \delta(t - nT_c). \quad (2.21)$$

---

<sup>1</sup> Since the implementation of a low-pass filter that actually stops all frequencies above a certain threshold is not possible, it is more correct to say that the effects of the aliasing problem are reduced to a level that does not disturb human perception. See [15] for a more extensive description of this issue.

The PIT can be expressed as a Fourier series:

$$p(t) = \frac{1}{T_c} \sum_{k=-\infty}^{\infty} e^{j\frac{2\pi}{T_c}kt} = \frac{1}{T_c} \sum_{k=-\infty}^{\infty} e^{j\Omega_{T_c}kt} \quad (2.22)$$

and  $s_p(t)$  can thus be reformulated as follows:

$$s_p(t) = \frac{s(t)}{T_c} \sum_{k=-\infty}^{\infty} e^{j\frac{2\pi}{T_c}kt} = \frac{s(t)}{T_c} \sum_{k=-\infty}^{\infty} e^{j\Omega_{T_c}kt}. \quad (2.23)$$

The FT of  $s_p(t)$  is thus:

$$S_p(\Omega) = \frac{1}{T_c} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} s(t) e^{j\Omega_{T_c}kt - j\Omega t} dt \quad (2.24)$$

and this can be interpreted as an infinite sum of shifted and scaled replicas of the FT of  $s(t)$ :

$$S_p(j\Omega) = \frac{1}{T_c} \sum_{k=-\infty}^{\infty} S_a(j(\Omega - k\Omega_{T_c})), \quad (2.25)$$

where each term of the sum is shifted by integer multiples of  $\Omega_{T_c}$  with respect to its neighbors.

The above situation is illustrated in Figure 2.6. The sampling induces replications of  $S_p(j\Omega)$  centered around integer multiples of  $\Omega_{T_c}$ , in correspondence of the impulses of the PIT Fourier transform. Each replication is  $2\Omega_{max}$  wide, where  $\Omega_{max} = 2\pi f_{max}$  is the highest angular frequency represented in the original signal  $s(t)$ . The  $k^{th}$  replication of  $S_p(j\Omega)$  stops at  $\Omega = k\Omega_{T_c} + \Omega_{max}$ , while the  $(k+1)$ th one starts at  $(k+1)\Omega_{T_c} - \Omega_{max}$ . The condition to avoid overlapping between consecutive replications is thus:

$$\Omega_{T_c} > 2\Omega_{max}. \quad (2.26)$$

Since  $\Omega = 2\pi f$ , Equation (2.26) corresponds to:

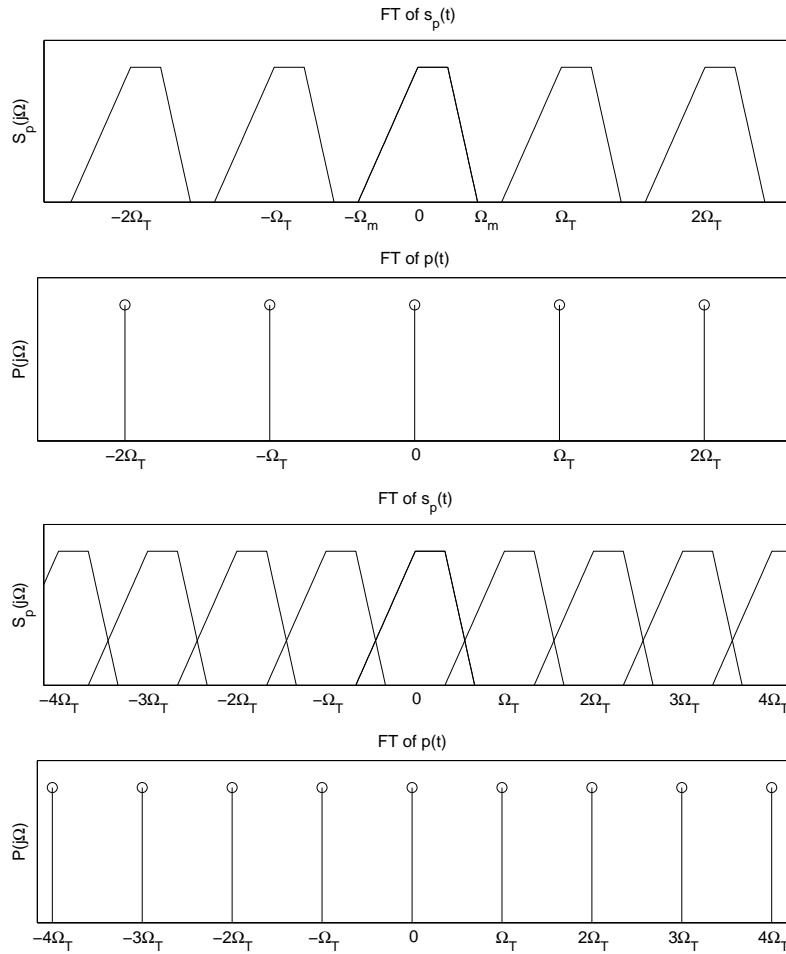
$$F > 2f_{max}. \quad (2.27)$$

This result is known as *sampling theorem*, and it is typically formulated as follows:

**Theorem 2.1.** *In order for a band-limited (i.e. one with a zero power spectrum for frequencies  $f > f_{max}$ ) baseband ( $f > 0$ ) signal to be reconstructed fully, it must be sampled at a rate  $F \geq 2f_{max}$ .*

Figure 2.6 shows what happens when the above condition is met (first and second plot from above) and when is not (third and fourth plot from above). Equation (2.26) is important because the overlapping between  $S_p(\Omega)$  replications is the frequency domain effect of the aliasing. In other words, the aliasing can be avoided if signals are sampled at a rate  $F$  higher or equal than the double of the highest frequency  $f_{max}$ .





**Fig. 2.6.** Sampling effect in the frequency domain. The first two plots from above show the sampling effect when  $\Omega_{T_c} > 2\Omega_m$ . The replications of  $S_p(j\Omega)$ , centered around the pulses in  $P(j\Omega)$ , are separated and the aliasing is avoided. In the third and fourth plot where the distance between the pulses in  $P(j\Omega)$  is lower than  $2\Omega_m$  and the aliasing takes place.

### 2.3.3 Linear Quantization

The second problem encountered in the acquisition process is the quantization, i.e. the approximation of a continuous interval of values by a relatively small set of discrete symbols or integer values. In fact, while the  $s[n]$  measures range, in general, in a continuous interval  $S = [-S_{max}, S_{max}]$ , only  $2^B$  discrete values are at disposition when  $B$  bits are available in a digital device. This section focuses on linear quantization methods, i.e. on quantization techniques that

split the  $s[n]$  range into  $2^B$  intervals and represent all the  $s[n]$  values lying in one of them with the same number. Other quantization techniques, called *vectorial*, will be described in Chapter 8.

The quantization can be thought of as a process that transforms a sequence of continuous values  $s[n]$  into a sequence of discrete values  $\hat{s}[n]$ . The most straightforward method to perform such a task is the so-called *linear pulse code modulation* (PCM) [27]. The PCM splits the interval  $S$  into  $2^B$  uniform intervals of length  $\Delta$ :

$$\Delta = \frac{S_{max}}{2^{B-1}}. \quad (2.28)$$

Each interval is given a code corresponding to one of the  $2^B$  numbers that can be described with  $B$  bits and  $\hat{s}[n]$  is obtained in one of the following ways:

$$\begin{aligned} \hat{s}[n] &= \text{sign}(c[n])\frac{\Delta}{2} + c[n]\Delta \\ \hat{s}[n] &= c[n]\Delta \end{aligned} \quad (2.29)$$

where  $c[n]$  is the code of the interval where  $s[n]$  falls. The two equations correspond to the situation depicted in left (*mid-riser quantizer*) and right (*mid-tread quantizer*) plots of Figure 2.7, respectively.

The use of  $\hat{s}[n]$  to represent  $s[n]$  introduces an error  $\epsilon[n] = s[n] - \hat{s}[n]$ . This leads to the use of the Signal to Noise Ratio (SNR) as a performance measure for quantization methods:

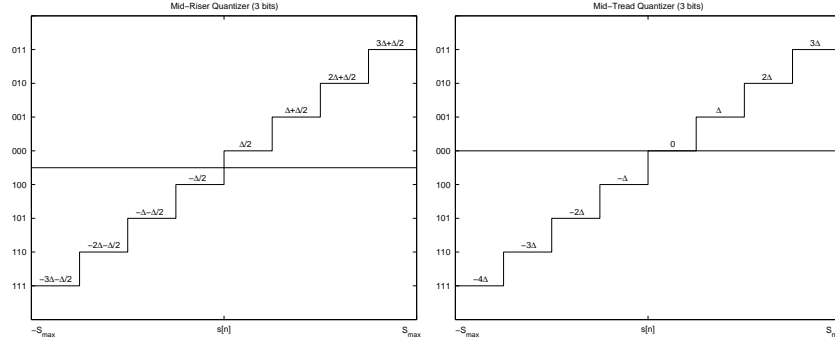
$$SNR = 10 \log_{10} \left\{ \frac{\sum_{n=0}^{M-1} s^2[n]}{\sum_{n=0}^{M-1} (s[n] - \hat{s}[n])^2} \right\} \quad (2.30)$$

where  $M$  is the number of samples in the data. Since  $\sum_n s^2[n]$  is the energy of a signal (see Section 2.5 for more details), the above equation is nothing but the ratio between the energy of the signal and the energy of the noise introduced by the quantization. The use of the logarithm (multiplied by 10) enables to use the dB as a measure unit (see Section 2.2). Higher SNR values correspond to better quantization performances because, for a given signal, the energy of the noise becomes smaller when the the values of the differences  $s[n] - \hat{s}[n]$  decrease.

The main limit of the SNR is that it might hide temporal variations of the performance. Local deteriorations can be better detected by using short term SNR measures extracted from segments of predefined length  $N$ . The average of local SNR values is called *segmental SNR* (SEGSNR) and it corresponds to the following expression:

$$SEGSNR = \frac{10}{L} \sum_{t=0}^{L-1} \log_{10} \left\{ \frac{\sum_{n=0}^{N-1} s^2[tN + n]}{\sum_{n=0}^{N-1} (s[tN + n] - \hat{s}[tN + n])^2} \right\} \quad (2.31)$$

where  $L$  is the number of  $N$  long segments spanning the  $M$  samples of the signal. The SEGSNR tends to penalize encoders with different performance for different signal energy and frequency ranges.



**Fig. 2.7.** Uniform quantization. The left plot shows a mid-riser quantizer, while the right plot shows a mid-tread quantizer.

In the case of the PCM, the upper bound of  $\epsilon[n]$  is  $\Delta$ ; in fact the maximum value that the difference  $s[n] - \hat{s}[n]$  can assume is the length of the interval where  $s[n]$  falls. The lower bound of the SNR is thus:

$$SNR_{PCM} = 10 \log_{10} \left\{ \frac{1}{\Delta^2} \sum_{n=0}^{M-1} s^2[n] \right\}. \quad (2.32)$$

The above expression shows the main limits of the PCM: if the SNR of lower energy signals decreases to a point that the perceptual quality of the quantized signal becomes unacceptable, the only way to improve the quantization performance is to reduce  $\Delta$ , i.e. to increase the number of bits  $B$ . On the other hand, it can happen that the same  $\Delta$  value that makes unacceptable the perceptual quality for lower-energy signals can be tolerated in the case of higher-energy sounds. For the latter, an increase of  $B$  is thus not necessary and it leads to an improvement of the SNR that goes beyond the human ear sensibility. This is not desirable, because the number of bits must be kept as low as possible in order to reduce the amount of memory necessary to store the data as well as the amount of bits that must be transmitted through a line.

The solutions proposed to address such a problem are based on the fact that the SNR is a ratio and can be kept constant by adapting the quantization error  $\epsilon[n]$  to the energy of the signal for any sample  $n$ . In other words, the SNR is kept at an acceptable level for all energy values by allowing higher quantization errors for higher-energy signals. Such an approach is used in differential PCM (DPCM), delta modulation (DM) and adaptive DPCM (ADPCM) [10]. However, satisfactory results can be obtained with two simple variants of the PCM that simply use a non uniform quantization interval. The variants, known as  $\mu$ -law and  $A$ -law PCM, are currently applied in telecommunications and are described in the next section.

### 2.3.4 Nonuniform Scalar Quantization

The previous section has shown that the SNR value can be kept constant at different energies by adapting the quantization error  $\epsilon[n]$  to the signal energy: the higher the energy of the signal, the higher the value of the quantization error that can be tolerated. This section shows how such a result can be obtained through functions called *logarithmic companders* and describes two quantization techniques based on such an approach and commonly applied in telecommunications:  $\mu$ -law and  $A$ -law PCM.

A logarithmic compander is a function that uses a logarithm to compress part of the domain where it is defined:

$$y[n] = \ln(|s[n]|) \text{sign}(s[n]), \quad (2.33)$$

where  $y[n] \in Y = [-\ln(S_{max}), \ln(S_{max})]$ ,  $\text{sign}(x) = 1$  when  $x \geq 0$  and  $\text{sign}(x) = -1$  when  $x < 0$  (see Section 2.3.3 for the meaning of symbols). If the uniform quantization is performed over  $Y$  (the vertical axis of Figure 2.8), then  $\hat{y}[n] - y[n] = \epsilon[n]$  and:

$$\hat{s}[n] = \exp(y[n]) \text{sign}(s[n]) = s[n] \exp(\epsilon[n]) \quad (2.34)$$

Since  $Y$  is quantized uniformly,  $\epsilon[n]$  can be approximated with the length  $\Delta_Y$  of the quantization interval. When  $\epsilon[n] \rightarrow 0$ , the above equation can be rewritten as follows using a Taylor series expansion:

$$\hat{s}[n] \simeq s[n](1 + \epsilon[n]) \quad (2.35)$$

and the expression of the SNR (see Equation (2.30)) for the logarithmic compander corresponds to

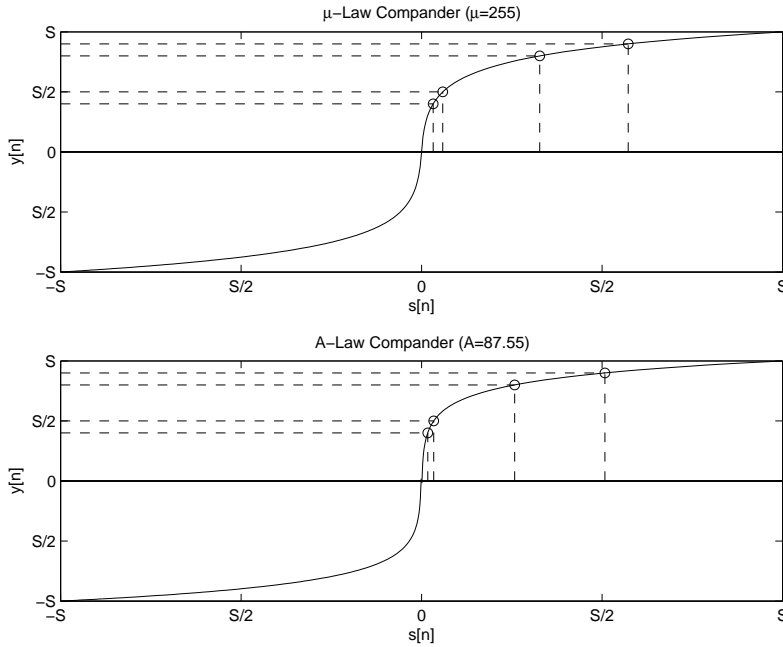
$$SNR_{log} = \sum_{n=0}^{M-1} \frac{1}{\Delta_Y^2} = \frac{M}{\Delta_Y^2}; \quad (2.36)$$

thus, for a given signal length,  $SNR_{log}$  does not depend on the energy. This happens because the uniform quantization of  $Y$  induces a nonuniform quantization on  $S$  such that the quantization step is proportional to the signal energy. When the energy of the signal increases, the quantization error is increased as well and the SNR of Equation (2.30) is kept constant.

The compander in Equation (2.33) brings to the above effect only when  $\epsilon[n] \rightarrow 0$ , but this is not possible for real applications. For this reason two variants are used in real applications<sup>2</sup>:

---

<sup>2</sup> There is no noticeable difference between the performance of the two companders, the  $A$ -law compander is used in Europe and other countries affiliated to the ITU (with  $A = 87.56$ ), while the  $\mu$ -law compander is mostly used in the USA (with  $\mu = 255$ ).



**Fig. 2.8.** Nonuniform quantization. The logarithmic companders induce finer quantization on lower-energy signals. Intervals with the same width on the vertical axis correspond to intervals with different width on the horizontal axis.

$$y[n] = S_{max} \frac{\log\left(1 + \mu \frac{|s[n]|}{S_{max}}\right)}{\log(1 + \mu)} \text{sign}(s[n]) \quad (2.37)$$

which is called the  $\mu$ -law and

$$y[n] = \begin{cases} S_{max} \frac{A \frac{|s[n]|}{S_{max}}}{1 + \log A} \text{sign}(s[n]); & 0 < \frac{|s[n]|}{S_{max}} < \frac{1}{A} \\ S_{max} \frac{1 + \log(A \frac{|s[n]|}{S_{max}})}{1 + \log A} \text{sign}(s[n]); & \frac{1}{A} < \frac{|s[n]|}{S_{max}} < 1 \end{cases} \quad (2.38)$$

which is called the  $A$ -law. It can be demonstrated that both above quantizers lead to an SNR independent of the signal energy.

In telephone communications, an SNR of around 35 dB is considered acceptable. While a uniform quantizer requires 12 bits to guarantee such an SNR all over the energy spectrum,  $A$ -law and  $\mu$ -law can achieve the same result by using only 8 bits [35]. For this reason, the above nonuniform quantization techniques are recommended by the *International Communications Union* and are applied to transmit speech through telephone networks [15].

## 2.4 Audio Encoding and Storage Formats

The number  $B$  of bits used to represent audio samples plays an important role in transmission and storage problems. In fact, the higher is  $B$ , the bigger is the amount of data to be transmitted through a channel and the larger is the memory space needed to store a recording. The amount of bits per time unit necessary to represent a signal is called *bit-rate* and it must be kept as low as possible to respect application constraints such as bandwidth and memory. On the other hand, a reduction of the bit-rate is likely to degrade the perceptual quality of the data and this, beyond a certain limit, is not tolerated by users (Section 2.3 shows that the reduction of  $B$  decreases the SNR of audio acquisition systems). The domain targeting techniques capable of reducing the bit-rate while still preserving a good perceptual quality is called *audio encoding*.

The main encoding methods result in audio *formats* (e.g. *MPEG*, *WAV*, *mp3*, etc.), i.e. into standardized ways of representing and organizing audio data inside files that can be used by computer applications. For this reason, this section presents not only encoding technologies, but also audio formats that make use of them. In particular, it will be shown how the development of new encoding methods and the definition of new formats is typically driven by two main factors: the first is the emergence of new applications that have bit-rate constraints tighter than the previous ones, the second is the expectation of users that accept different perceptual qualities depending on the applications.

The encoding problem is the subject of monographies [5] and tutorials [29][35] that provide extensive introductions to the different algorithms and formats. For the MPEG audio format and coding technique, both tutorial level [4][7][26] articles and monographies [21] are available.

### 2.4.1 Linear PCM and Compact Discs

The earliest encoding approach is the linear PCM presented in Section 2.3. Although simple, such a technique is the most expensive in terms of bit-rate (see below) and the most effective for what concerns perceptual quality. Since it reproduces the whole information contained in the original waveform, the linear PCM is said *lossless*, in opposition to *lossy* approaches that discard selectively part of the original signal (see the rest of this section for more detail). In general, the samples are represented with  $B = 16$  bits because this makes the quantization error small enough to be inaudible even by trained listeners (the so-called *golden ears* [29]). The sampling frequency commonly used for high-fidelity audio is  $F = 44.1$  kHz and this leads to a bit rate of  $2BF = 1,411,200$  bits per second. The factor 2 accounts for the two aural channels in a stereo recording.

Although high, such a bit-rate could be accommodated on the first supports capable of storing digital audio signals, i.e. digital audio tapes (DAT) and compact discs (CD). These last in particular started to spread in the early

eighties, although invented in the sixties, and they are now, together with CD players, some of the most important consumer electronic products. One hour of high fidelity stereo sound at the 16-bit PCM rate requires roughly 635 MB. A CD can actually store around 750 MB, but the difference is needed for *error correction bits*, i.e. data required to recover acquisition errors. Since CDs have been used mainly to replace old vinyl recordings that were often shorter, the one-hour limit was largely accepted by users, and still is. For this reason, there was no pressure to decrease the PCM bit-rate in order to store more sound on CDs. At the same time, the perceptual improvement determined by the use of digital rather than analogic supports was so high, that the user expectations increased significantly and the CD-quality is currently used as a reference for any other encoding technique [26].

The linear PCM is the basis for several other formats that are used in conditions where the memory space is not a major problem: Windows *WAV*, Apple *AIFF* and Sun *AU*. In fact, such formats, with different values of  $B$  and  $F$ , are used to store sound on hard disks that are today large enough to contain hours of recordings and that promise to grow at a rate that makes the space constraint marginal.

The same does not apply to telephone communications where a high bit-rate results into an ineffective use of the lines. For this reason, the first efforts in reducing the bit-rate came from that domain. On the other hand, the development of encoding techniques for phone communications has an important advantage: since consumers are used to the fact that the so-called *telephone speech* is not as natural as in other applications (e.g. radio and television), their expectations are significantly lower and the bit-rate can be reduced with simple modifications of the linear PCM.

Section 2.3 shows that the main limit of the linear PCM is that the quantization error does not change with the signal energy. In this way, the parameter  $B$  must be kept at a level that leads to an SNR acceptable at low energies, but high beyond human hearing sensibility at higher energies. In other words, there is a waste of bits at higher energies. The  $A$ -law and  $\mu$ -law logarithmic companders address such a problem by adapting the quantization errors to the amplitude of the signals and reduce by roughly one third the bit-rate necessary to achieve a certain perceptual quality. For this reason the logarithmic companders are currently advised by the *International Telecommunications Union* (ITU) and are widely applied with  $A = 87.55$  and  $\mu = 255$ .

One of the most important lessons in the phone case, is that user expectations are not directed towards the highest possible quality, but simply at keeping constant the perceptual level in a given application. For this reason, the performance of an encoder is measured not only with the SNR, but also with the *mean opinion score* (MOS), a subjective test involving several *naïve* listeners, i.e. people that do not know encoding technologies (this might bias their evaluations). Each listener is asked to give a score between 1 (bad) and 5 (excellent) to a given encoded sound and the resulting MOS value is the average of all judgments given by the assessors. An MOS of 4.0 or more defines

*good* or *toll* quality where the encoded signal cannot be distinguished from the original one. An MOS between 3.5 and 4.0 is considered acceptable for telephone communications [15]. The test can be performed informally, but the results are accepted in the official organizations only if they respect the rigorous protocols given by the ITU [1].

#### 2.4.2 MPEG Digital Audio Coding

Logarithmic companders and other approaches based on the adaptation of the noise to the signal energy (see Section 2.3) obtain significant reductions of the bit-rate. However, these are not sufficient to respect bandwidth and space constraints imposed by applications developed in the last years. Multimedia, streaming, online applications, content diffusion on cellular phones, wireless transmission, etc. require to go beyond the reduction by one-third achieved with *A-law* and  *$\mu$ -law* encoding techniques. Moreover, user expectations correspond now to CD-like quality and any degradation with respect to such a perceptual level would not be accepted. For this reason, several efforts were made in the last decade to improve encoding approaches.

*MPEG* is the standard for multimedia (see Chapter 3), its digital audio coding technique is one of the major results in audio coding and it involves several major changes with respect to the linear PCM. The first is that the *MPEG* architecture is organized in *Layers* containing sets of algorithms of increasing complexity. Table 2.1 shows the bit-rates achieved at each layer and the corresponding compression rates with respect to the 16-bit linear PCM.

The second important change is the application of an *analysis and synthesis* approach implemented in layers I and II. This consists in representing the incoming signals with a set of compact parameters, in the case of sound frequencies, which can be extracted in the encoding phase and used to reconstruct the signal in the following decoding step (for a detailed description of the algorithms of the first two layers, see [29]). An average MOS of 4.7 and 4.8 has been reported for monaural layer I and II codecs operating at 192 and 128 kbits/sec [25].

**Table 2.1.** MPEG audio layers. This table reports bit-rates (central column) and compression rates (right column), compared to CD bit-rate, achieved at different layers in the MPEG coding architecture. The compression rate is the ratio between CD and MPEG bit-rate at the same audio quality level.

| Layer | Bit-rate   | Compression |
|-------|------------|-------------|
| I     | 384 kb/sec | 4           |
| II    | 192 kb/sec | 8           |
| III   | 128 kb/sec | 12          |



The third major novelty is the application of psychoacoustic principles capable of identifying and discarding *perceptually irrelevant* frequencies in the signal. By perceptually irrelevant it is meant that a frequency cannot be perceived by human ears even if it is present in the signal, thus it can be discarded without degradation of the perceptual quality. Such an approach is called *perceptual coding* and, since part of the original signal is removed, the encoding approach is defined *lossy*. The application of the psychoacoustic principles is performed at layer III and it reduces by 12 the bit-rate of the linear PCM while achieving an average MOS between 3.1 and 3.7 [25]. The *MPEG* layer III is commonly called *mp3* and it is used extensively on the web because of its high compression rate (see Table 2.1). In fact, the good tradeoff between perceptual quality and size makes the *mp3* files easy to download and exchange. The format is now so popular that it gives the name to a new class of products, i.e. the *mp3 players*.

The main improvements of the *mp3* with respect to previous formats come from the application of perceptual coding. Section 2.4.4 provides a description of the main psychoacoustic phenomena used in *mp3*.

### 2.4.3 AAC Digital Audio Coding

The acronym AAC stands for *advanced audio coding* and the corresponding encoding technique is considered as the natural successor of the *mp3* (see the previous section) [29]. The structures of *mp3* and AAC are similar, but the latter improves some of the algorithms included in the different layers.

AAC contains two major improvements with respect to *mp3*. The first is the higher adaptivity with respect to the characteristics of the audio. Different analysis windows (see Section 2.5) are used when the incoming sound has frequencies concentrated in a narrow interval or when strong components are separated by more than 220 Hz. The result is that the perceptual coding gain is maximized, i.e. most of the bits are allocated for perceptually relevant sound parts. The second improvement is the use of a predictor for the quantized spectrum. Some audio signals are relatively stationary and the same spectrum can be used for subsequent analysis frames (see Section 2.5). When several contiguous frames use the same spectrum, this must be encoded only the first time and, as a consequence, the bit-rate is reduced. The predictor is capable of deciding in advance whether the next frame requires to compute a new spectrum or not.

In order to serve different needs, the AAC provides three profiles of decreasing complexity: the main profile offers the highest quality, the low-complexity profile does not include the predictor and the sampling-rate-scaleable profile has the lowest complexity (see [26] for details about each profile). The main profile AAC has shown higher performance the other formats in several comparisons<sup>3</sup>: at a bit-rate of 128 kb/sec, listeners cannot distinguish between

---

<sup>3</sup> The results can be found on [www.apple.com/quicktime/technologies/aac/](http://www.apple.com/quicktime/technologies/aac/).

original and coded stereo sound. If the bit-rate is decreased at 96 kb/sec, AAC has a quality higher than mp3 at 128 kb/sec. On the other hand, if both AAC and mp3 have a bit-rate of 128 kb/sec, the AAC shows a significantly superior performance.

#### 2.4.4 Perceptual Coding

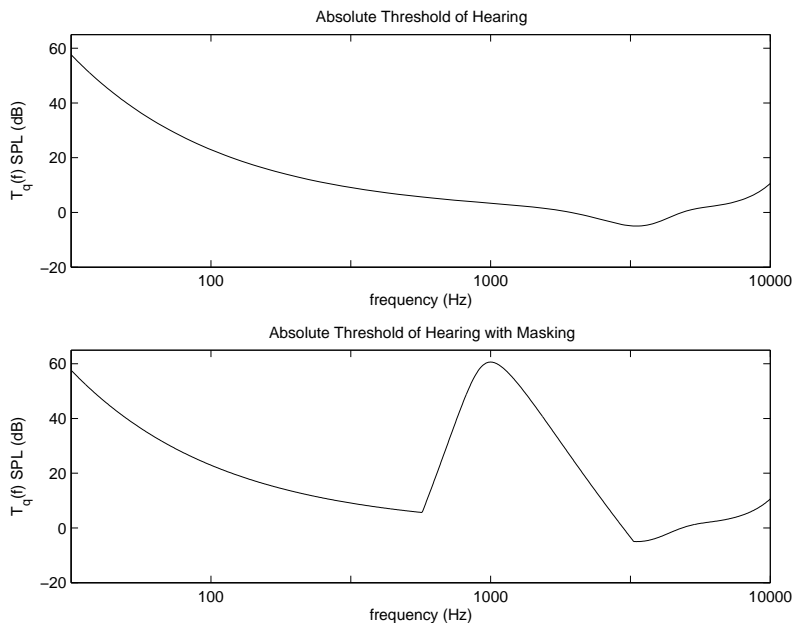
The main issue in perceptual coding is the identification of the frequencies that must be coded to preserve perceptual quality or, conversely, of the frequencies that can be discarded and for which no bits must be allocated. The selection, in both above senses, is based on three psychoacoustic phenomena: the existence of critical bands, the absolute threshold of hearing (TOH) and the masking. Critical band analysis has been introduced at the end of Section 2.2, the other two phenomena are briefly described in the following.

Section 2.2 defines the TOH as the lowest energy that a signal must carry to be heard by humans (corresponding to an intensity  $I_0 = 10^{12}$  Watts per square meter). This suggests as a first frequency removal criterion that any spectral component with an energy lower than the TOH should not be coded. However, perceptual experiments have shown that the above TOH does not apply to any frequency and that the minimum audible energy is a function of  $f$  [12]:

$$T_q(f) = 3.64 \left( \frac{f}{10^3} \right)^{-0.8} - 6.5e^{-0.6\left(\frac{f}{10^3} - 3.3\right)^2} + 10^{-3} \left( \frac{f}{10^3} \right)^4 \quad (dB SPL). \quad (2.39)$$

The function  $T_q(f)$  is referred to as *absolute* TOH and it enables to achieve better bit-rate reduction by removing any spectral component with energy  $E_0 < T_q(f_0)$ . Absolute TOH is plotted in Figure 2.9, the lowest energy values correspond to frequencies ranging between 50 and 4000 Hz, not surprisingly those that propagate better through the middle ear (see Section 2.2). The main limit of the  $T_q(f)$  introduced above is that it applies only to pure tones in noiseless environments, while sounds in everyday life have a more complex structure. In principle, it is possible to decompose any complex signal into a sum of waves with a single frequency  $f_0$  and to remove those with energy lower than  $T_q(f_0)$ , but this does not take into account the fact that the perception of different frequencies is not independent.

In particular, components with a certain frequency can stop the perception of other frequencies in the auditory system. Such an effect is called *masking* and it modifies significantly the curve in Figure 2.9. The waves with a given frequency  $f$  excite the auditory nerves in the region where they reach their maximum amplitude (the nerves are connected to the cochlea walls). When two waves of similar frequency occur together and their frequency is around the center of a critical band (see Section 2.2), the excitation induced by one of them can prevent from hearing the other. In other words, one of the two sounds (called *masker*) masks the other one (called *maskee*). From an encoding point



**Fig. 2.9.** Absolute TOH. The TOH is plotted on a logarithmic scale and shows how the energy necessary to hear frequencies between 50 and 4000 kHz is significantly lower than the energy needed for other frequencies.

of view, this is important because no bits accounting for maskee frequencies need to be allocated in order to preserve good perceptual quality. The inclusion of masking in audio encoding is a complex process (see [29] for a detailed description for application in MPEG coding). For the sake of simplicity, we will show only how masker and maskee frequencies are identified in the two most common cases: tone masking noise (TMN) and noise masking tone (NMT).

The first step is to find tone and noise frequencies. The  $f$  values corresponding to masker tones are identified as peaks in the power spectrum with a difference of at least 7 Barks with respect to neighboring peaks. Noise maskers are detected through the geometric mean of frequencies represented between to consecutives tonal maskers. TMN takes place when noise masks tones with lower energy. Empirical models show that this happens when the difference between tone and noise energies is below a threshold  $T_T(b)$  that can be calculated as follows:

$$T_T(b) = E_N - 6.025 - 0.275 \cdot g + S_m(b - g) \quad (2.40)$$

where  $b$  and  $g$  are the Bark frequencies of tone and noise, respectively,  $E_N$  is the noise energy and  $S_m(h)$  is the *spread of masking* function given by

$$S_m(h) = 15.81 + 7.5 \cdot (h + 0.474) - 17.5 \sqrt{1 + (h + 0.474)^2} \quad (2.41)$$

where  $h$  is the Bark frequency difference between noise and tone. The expression of the threshold for the NMT is similar:

$$T_N(b) = E_T - 2.025 - 0.175 \cdot g + S_m(b - g) \quad (2.42)$$

where  $E_T$  is the tone energy. Although Equations (2.40) and (2.42) seem to be symmetric, there is an important difference between TMN and NMT: in the first case only tones with signal-to-mask ratio (SMR) between -5 and 5 dB can be masked, while in the second case the SMR range where the masking takes place is between 21 and 28 dB. A tone can thus mask noise with energies roughly 100 to 1,000 times higher, while a noise can mask tones with energies from around one-third to three times its energy. The lower plot in Figure 2.9 shows the effect of a masking tone noise of frequency 1 kHz and energy 69 dB. The energy necessary to hear frequencies close to 1 kHz is significantly higher than the corresponding TOH and this enables to reduce the number of bits necessary to encode the frequency region where masking takes place.

## 2.5 Time-Domain Audio Processing

The result of the acquisition process is a sequence of quantized physical measures  $\{s[n]\} = (s[1], s[2], \dots, s[N])$ . Since both  $n$  and  $s[n]$  are discrete, such sequences are referred to as *digital signals* and their form is particularly suitable for computer processing. This section presents some techniques that extract useful information from the analysis of the variations across the sequences. The corpus of such techniques is called *time-domain audio processing* in opposition to *frequency-domain* techniques which operate on frequency distributions (see Appendix B for more details).

After presenting the fundamental notion of *system* and related properties, the rest of this section focuses on how to extract information related to energy and frequency. The subject of this section is covered in more detail in several speech and signal processing texts [15][22][33].

### 2.5.1 Linear and Time-Invariant Systems

Any operator  $T$  mapping a sequence  $s[n]$  into another digital signal  $y[n]$  is called *discrete-time system*:

$$y[n] = T\{s[n]\}, \quad (2.43)$$

the element  $y[n]$  is a function of a single sample  $s[n]$ , of a subset of the samples of  $\{s[n]\}$  or of the whole input digital signal  $\{s[n]\}$ . In the following, we show three examples corresponding to each of these situations: The *ideal delay* (function of a single sample), the *moving average* (function of a subset), and the *convolution* (function of the whole signal).

The *ideal delay* system is as follows:

$$y[n] = s[n - n_0] \quad (2.44)$$

where  $n_0$  is an integer constant and  $y[n]$  is function of the the only sample  $s[n - n_0]$ . The *moving average* is:

$$y[n] = \frac{1}{K_1 + K_2 + 1} \sum_{k=-K_1}^{K_2} s[k] \quad (2.45)$$

where  $K_1$  and  $K_2$  are two integer constants and  $y[n] = T\{s[n]\}$  is function of the samples in the interval between  $n - K_2$  and  $n + K_1$ . The expression of the convolution is:

$$y[n] = \sum_{k=-\infty}^{\infty} s[k]w[n - k] \quad (2.46)$$

where  $w[n]$  is another digital signal and  $y[n]$  is a function of the whole sequence  $\{s[n]\}$ .

A system is said *linear* when it has the following properties:

$$\begin{aligned} T\{s_1[n] + s_2[n]\} &= T\{s_1[n]\} + T\{s_2[n]\} \\ T\{as[n]\} &= aT\{s[n]\} \end{aligned} \quad (2.47)$$

where  $s_1[n]$  and  $s_2[n]$  are two different digital signals and  $a$  is a constant. The first property is called *additivity* and the second *homogeneity* or *scaling*. The two properties can be combined into the so-called *superposition principle*:

$$T\{as_1[n] + bs_2[n]\} = aT\{s_1[n]\} + bT\{s_2[n]\}. \quad (2.48)$$

Given a signal  $\hat{s}[n] = s[n - n_0]$ , a system is said to be *time invariant* when:

$$\hat{y}[n] = T\{\hat{s}[n]\} = y[n - n_0]. \quad (2.49)$$

The above equation means that a shift of the origin in the input digital signal determines the same shift in the output sequence. In other words, the effect of the system at a certain point of the sequence does not depend on the sample where  $T$  starts to operate.

When a system is LTI, i.e. both linear and time-invariant, the output sequence  $y[n]$  can be obtained in a peculiar way. Consider the so-called *impulse*, i.e. a digital signal  $\delta[n]$  such that  $\delta[n] = 1$  for  $k = 0$  and  $\delta[n] = 0$  otherwise, the output of a system can be written as follows:

$$y[n] = T \left\{ \sum_{k=-\infty}^{\infty} s[k]\delta[n - k] \right\} = \sum_{k=-\infty}^{\infty} s[k]T\{\delta[n - k]\}, \quad (2.50)$$

and the above equation can be rewritten as:

$$y[n] = \sum_{k=-\infty}^{\infty} s[k]h[n - k] \quad (2.51)$$

which corresponds to the convolution between the input signal  $s[n]$  and  $h[n - k]$ , i.e. the response of the system to an impulse at time  $n$ . As a consequence, an LTI system is completely determined by its impulse response  $h[n]$ , in the sense that  $h[n]$  can be used to obtain  $y[n]$  for any other input signal  $s[n]$  through a convolution operation  $s[n] * h[n]$ .<sup>4</sup>

### 2.5.2 Short-Term Analysis

Figure 2.11 shows a speech waveform sampled at 8 kHz. Such a value of  $F$  is common for spoken data because the highest formant frequencies in the human voice are around 4 kHz (see Section 2.2) and the lowest point of the absolute TOH curve for the human auditory system corresponds roughly to such frequency (see Figure 2.9). Speech data are thus low-pass filtered at 4 kHz and sampled at 8 KHz to meet the sampling theorem conditions. The waveform of Figure 2.11 shows two important aspects: the first is that different segments of the signal have different properties (e.g. speech and silence), the second is that the signal properties change relatively slowly, i.e. they are stable if an interval short enough is taken into account (e.g. 20 – 30 ms). Such assumptions underly the *short-term analysis*, an approach which takes into account segments short enough to be considered as sustained sounds with stable properties.

In mathematical terms this means that the value of the property  $Q[n]$  at time  $nT$ , where  $T = 1/F$  is the sampling period, can be expressed as follows:

$$Q[n] = \sum_{m=-\infty}^{\infty} K(s[m])w[n - m] \quad (2.52)$$

where  $K$  is a transform, either linear or nonlinear, possibly dependent upon a set of adjustable parameters, and  $w[n]$  is the so-called analysis *window*. Two analysis windows are commonly applied: the first is called *rectangular* and the second is called *Hamming*. The latter has been introduced to avoid the main problems determined by the rectangular window, i.e. the presence of too high secondary lobes in the Fourier transform (see Appendix B). The *rectangular* window is defined as follows:

$$w[n] = \begin{cases} 1 & : 0 \leq l \leq N - 1 \\ 0 & : l < 0 \\ 0 & : l \geq N \end{cases}$$

and the Hamming window:

---

<sup>4</sup> The advantages of this property are particularly evident in the frequency domain. In fact, the Fourier transform of a convolution between two signals corresponds to the product between the Fourier transforms of the single signals, and this simplifies significantly the analysis of the effect of a system in the frequency domain.

$$w[n] = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) & : 0 \leq l \leq N - 1 \\ 0 & : l < 0 \\ 0 & : l \geq N. \end{cases}$$

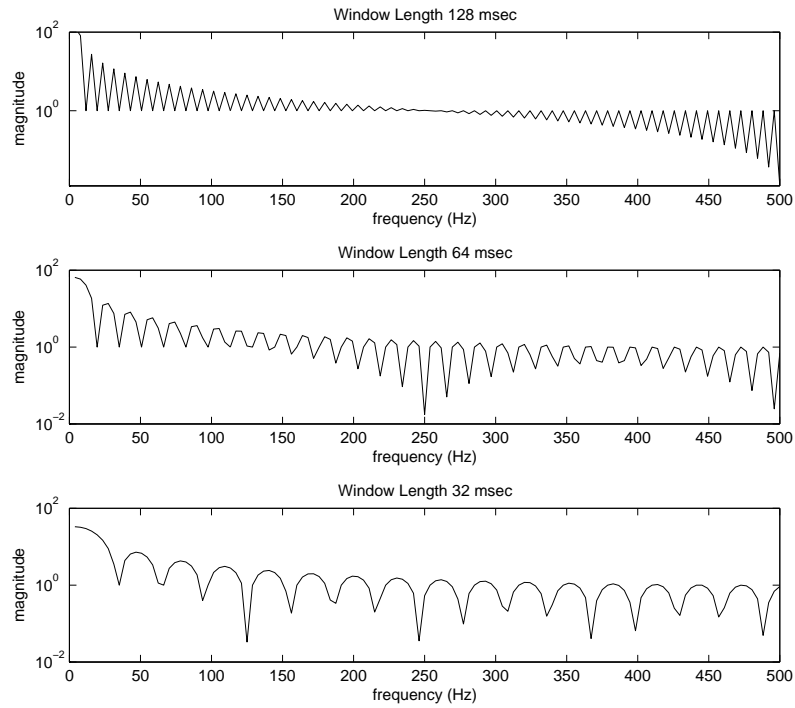
In both above cases, as well as for any finite window, it is necessary to set the parameter  $N$ , the so-called *window length*. The value of  $N$  must be the tradeoff between two conflicting requirements: the first is that the window must be short enough to detect rapid changes of  $Q$ , the second is that it must be long enough to smooth local random fluctuations. Moreover, no window length gives satisfactory results for every application and different choices must be made for different tasks. In the case of spoken data, it is common to have a window corresponding to few fundamental periods  $T0 = 1/F0$ , where  $F0$  is the fundamental frequency (see Section 2.2). In more general terms, the problem is addressed by observing that the variations of  $Q$  can be studied through the Fourier transform (FT) of  $Q[n]$  (the unexperienced reader can move directly to Section 2.5.3). In this case high frequencies in the spectrum correspond to rapid  $Q$  variations, while low frequencies components are due to slow changes.

Since Equation (2.52) can be interpreted as a discrete convolution, the FT of  $Q[n]$  can be obtained as a product of the FT's of  $K(s[n])$  and  $w[n]$ . The effect of  $N$  on the frequency with which  $Q$  changes can thus be evaluated through the FT of the window. Figure 2.10 shows the spectra of rectangular windows of different length. The windows act as a low-pass filters with cutoff frequencies  $f_r = F/N$  ( $f_h = 2F/N$  for the Hamming windows). The consequence is that the longer is the window, the narrower is the band of accepted frequencies. In other words, long windows tend to mask rapid changes and vice versa for short windows. In speech recognition (see Chapter 12) the window is typically 10-30 ms long. The reason is that physiological measurements performed using X-rays have shown that during such a time humans cannot significantly change the shape of the vocal tract.

### 2.5.3 Time-Domain Measures

This section presents the most important properties that can be extracted from a signal in the time domain. All of the properties are obtained with a short-term approach and provide a rough but meaningful representation of the audio signals (particular attention will be paid to speech data).

The first two properties are short-time *energy* and *average magnitude*. They carry the same kind of information, but the second one is less sensitive to local fluctuations. They are especially important to detect silences or to distinguish between voiced and unvoiced segments in spoken data, but they also play a role for the reduction of the bit-rate during the quantization. In fact, higher quantization errors can be allowed for higher energy signals (see Section 2.3). The short-time energy  $E[n]$  of a signal can be extracted through the following convolution:



**Fig. 2.10.** Window effect in the frequency domain. The three plots show the spectrum of rectangular windows of length 128, 64 and 32 ms, respectively. All spectra show a first minimum in correspondence of  $f_r = F/\Delta t$  Hz, where  $\Delta t$  is the length of the window. This means that variations of frequency higher than  $f_r$  are filtered and that longer windows tend to smooth higher frequency variations (and vice versa).

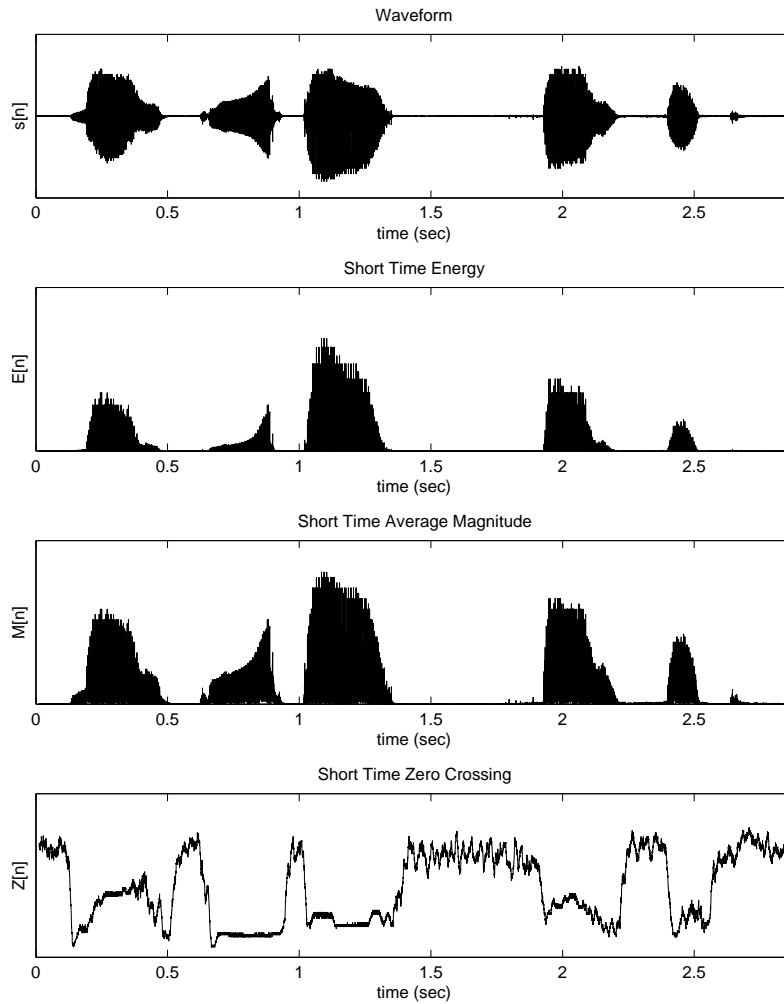
$$E[n] = \sum_{m=-\infty}^{\infty} s^2[n]w[n-m]. \quad (2.53)$$

The use of the square makes  $E[n]$  too sensitive to the highest values of  $s[n]$  that can be due to local random fluctuations. Moreover, the lowest energy parts of the signal tend to be suppressed as it can be observed in Figure 2.11: the energy of the unvoiced phonemes at the end of the word *six* is so much lower than the other parts of the words that it can be difficult to distinguish them with respect to the silence. For this reason,  $E[n]$  is often replaced with the short-term average magnitude  $M[n]$ :

$$M[n] = \sum_{m=-\infty}^{\infty} |s[n]w[n-m]|. \quad (2.54)$$

The dynamic range of  $M[n]$  is smaller and the differences are smoother than in the  $E[n]$  case. This can be seen at the end of the word *six* in Figure 2.11





**Fig. 2.11.** Time domain processing. The plots show (from the top to the bottom) a waveform, the short-time energy, the short-time average magnitude, the short-time average zero crossing rate. The sampling rate is 8000 Hz and the window is  $12.5 \mu\text{s}$  long.

where the unvoiced phonemes have an average magnitude lower, but still comparable with the  $M[n]$  value of voiced phonemes.

The length of the window should correspond more or less to a pitch period (see Section 2.2). Shorter windows detect uninteresting local fluctuations, while longer windows miss changes that should not be neglected. Since the pitch of human voices ranges between 50 (for male voices) and 400 kHz (for small children and women), no window length is optimal for any case, How-

ever, satisfactory results can be achieved, on average, with a 20-30 ms long analysis frame. Energy and magnitude are often used as features in speech recognition systems [15] as well as in multimedia content analysis where they have been applied to detect emotional states [18], to identify audio segments likely to attract the attention [20], to perform affective analysis [14].

Another important aspect of a signal is the frequency content. This is typically obtained through the Fourier transform (see Appendix B), but a simple time domain measure, called *short time average zero-crossing rate* ZCR, enables us to obtain a rough idea of the frequencies represented in the data. Such a measure can be obtained as follows:

$$Z[n] = \frac{1}{2N} \sum_{m=-\infty}^{\infty} |\text{sign}(s[m]) - \text{sign}(s[m-1])|w[n-m] \quad (2.55)$$

where  $w(l)$  is a rectangular window of length  $N$ . If  $s(t)$  is a sinusoid of frequency  $f$ , then there are two zero crossings every  $T$  seconds, where  $T = 1/f$ . If  $s(t)$  is sampled at a rate  $F > 2f$  for a time  $\Delta t$  corresponding to a high multiple of  $T$ , the average number of zero crossings  $Z$  can be obtained as follows:

$$Z \simeq \frac{2f}{F} \quad (2.56)$$

where  $f/F$  is nothing else than the number of sinusoid cycles per sampling period. For this reason,  $Z[n]$  provides a rough description of the frequency content in  $s[n]$ . The lowest plot of Figure 2.11 shows the value of  $Z[n]$  for the spoken utterance used as example so far: on average, the  $Z[n]$  value is between 0.1 and 0.2 in the spoken segments and this corresponds, using Equation (2.56), to frequencies between 400 and 800 Hz. This is compatible with the fact that the speaker is a woman (and the fundamental frequencies are up to 300 Hz for women) and with the fact that the energy of the speech tends to concentrate below 3000 Hz. The value of  $Z[n]$  in the silence segments is, on average, between 0.5 and 0.6 and this accounts for frequencies between 2000 and 2400 Hz. The reason is that the energy of nonspeech segments is concentrated on high-frequency noise. However, the above frequencies values must be considered indicative and must be used to discriminate rather than to describe different segments. The ZCR has been used in several audio processing technologies including the detection of word boundaries [32], speech-music discrimination [8][34], audio classification [19].

The property examined next is the *autocorrelation function*  $\phi[k]$  which has a different expression depending on the kind of signal under examination. For *finite energy* signals  $\phi[k]$  is defined as follows:

$$\phi[k] = \sum_{m=-\infty}^{\infty} s[m]s[m+k]. \quad (2.57)$$

A signal is said to be finite energy when the following sum is finite:

$$E = \sum_{n=-\infty}^{\infty} s^2[n]. \quad (2.58)$$

for *constant power* signals the expression is:

$$\phi[k] = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{m=-N}^N s[m]s[m+k]. \quad (2.59)$$

A signal is said to be constant power when the following sum is constant:

$$P = \sum_{n=-T}^T s^2[n_0 - n] \quad (2.60)$$

for any  $n_0$  and  $T$ .  $P$  can be thought of as the signal power, i.e. the average signal energy per time unit. The autocorrelation function has several important properties. The first is that if  $s[n] = s[n+mp]$ , where  $m$  is an integer number, then  $\phi[k] = \phi[k+mp]$ . In other words, the autocorrelation function of a periodic signal is periodic with the same period. The second is that  $\phi[k] = \phi[-k]$ , i.e. the autocorrelation function is even and it attains its maximum for  $k = 0$ :

$$|\phi[k]| \leq \phi[0] \quad \forall k. \quad (2.61)$$

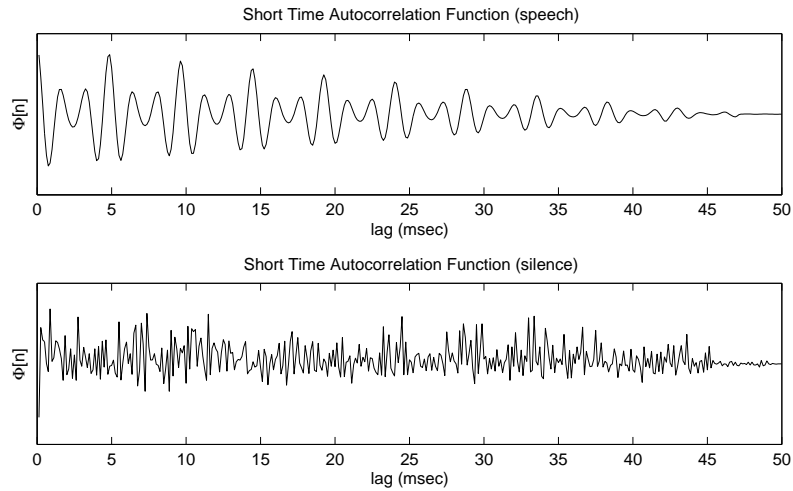
The value of  $\phi[0]$  corresponds to the total energy of the signal which is thus a particular case of the autocorrelation function.

Equation (2.57) is valid for the signal as a whole, but in audio processing the analysis is performed, in general, on an analysis frame. This requires the definition of a *short-term autocorrelation function*:

$$R_n[k] = \sum_{m=-\infty}^{\infty} s[m]w[n-m]s[m+k]w[n-m-k]. \quad (2.62)$$

Such an expression corresponds to the value of  $\phi[k]$  calculated over the intersection of two windows shifted by  $k$  sampling periods with respect to each other. If  $k > N$  (where  $N$  is the window length), then  $R_n[k] = 0$  because there is no intersection between the two windows.

The short-term properties considered so far (energy, average magnitude and average ZCR) provide a single value for each analysis frame identified by a specific position of the window. This is not the case of the short-time autocorrelation function which provides, for each analysis frame, a function of the *lag*. Figure 2.12 shows the short-term autocorrelation function obtained from a window of length  $N = 401$  (corresponding to 50 ms). Upper and lower plots have been obtained over a speech ( $t = 1.2$  sec. in Figure 2.11) and a silence segment ( $t = 1.5$  sec. in Figure 2.11) respectively. In the first case there are clear peaks appearing roughly every 5 msec, and this corresponds to a fundamental frequency of around 200 Hz. In the second case no periodicity



**Fig. 2.12.** Short term autocorrelation function. Upper and lower plots show the short term autocorrelation function for a speech and a silence point respectively. The plot in the silence case does not show any periodicity, while in the speech case there are peaks appearing roughly every 5 msec. This corresponds to a fundamental frequency of around 200 Hz, a value compatible with the ZCR measures made over the same signal and with the fact that the speaker is a woman.

is observed and  $R_n[k]$  looks rather like a high-frequency noise-like waveform. The autocorrelation function can thus be used as a further description of the frequency content that can help in discriminating different parts of the signal. Figure 2.12 shows that the amplitude of  $R_n[k]$  decreases as the lag increases. The reason is that for higher values of  $k$  the intersection between the two windows decreases and there are less addends in the sum of Equation (2.62).

The autocorrelation function has been used to detect the music meter [6], pitch detection [31], music and audio retrieval [13][37], audio fingerprinting [36], and so on.

## Problems

**2.1.** Consider a sound of intensity  $I = 5 \text{ dB}$ . Calculate the energy emitted by its source in a time interval of length  $\Delta t = 22.1 \text{ s}$ . Given the air acoustic impedance  $Z = 410 \text{ Pa} \cdot \text{s} \cdot \text{m}^{-1}$ , calculate the pressure corresponding to the maximum compression determined by the same sound wave.

**2.2.** Human ears are particularly sensitive to frequencies between 50 and 4000 Hz. Given the speed of sound in air ( $v \simeq 331.4 \text{ m} \cdot \text{s}^{-1}$ ), calculate the wavelengths corresponding to such frequencies.

**2.3.** Consider a sum of  $N$  sinusoids with frequencies  $f_0, 3f_0, \dots, (2N + 1)f_0$ :

$$f(t) = \sum_{n=0}^N \frac{1}{2n+1} \sin[2\pi f_0(2n+1)t] \quad (2.63)$$

Plot  $f(t)$  in the range  $[0, 10]$  for  $f_0 = 1$  and  $N = 1, 2, \dots, 100$  and observe the signal  $f(t)$  converges to.

**2.4.** The Mel scale (see Section 2.2.3) maps frequencies  $f$  into values  $B(f)$  that are more meaningful from a perceptual point of view. Segment the  $B(f)$  interval  $[0, 3375]$  into 20 intervals of the same length and find the frequencies  $f$  corresponding to their limits.

**2.5.** Extract the waveform from an audio file using *HTK* (see Chapter 12 for a description of the HTK software package) and calculate the number of bits  $N$  necessary to represent the sample values. Perform a uniform quantization of the waveform using a number of bits  $n$  ranging from 2 to  $N - 1$  and calculate, for each  $n$ , the signal-to-noise ratio (SNR). Plot the SNR as a function of  $n$ .

**2.6.** Calculate sampling frequency and bit-rate of the audio file used in Problem 2.5.

**2.7.** Plot the TOH in presence of a masking tone noise of frequency 200 Hz and intensity 50 dB.

**2.8.** Consider the system known as *moving average* (see Section 2.5). Demonstrate that such system is linear and time invariant.

**2.9.** Consider an audio file including both speech and silence and extract the waveform it contains. Obtain magnitude and zero crossing rate as a function of time using a rectangular analysis window 30 ms long. A pair  $(M[n], Z[n])$  is available for each sample  $s[n]$  and can be plotted on a plane where the axes are magnitude and ZCR. Do sound and speech samples form separate clusters (see Chapter 6)?

**2.10.** Demonstrate that the autocorrelation function  $R_n[k]$  corresponds to the short time energy when  $k = 0$  and that  $|R_n[k]| < R_n[0]$  for  $k > 0$ .



---

## References

1. Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems. Technical report, International Telecommunication Union, 1997.
2. L.L. Beranek. Concert hall acoustics. *Journal of the Acoustical Society of America*, 92(1), 1992.
3. D.T. Blackstock. *Fundamentals of Physical Acoustics*. John Wiley and Sons, 2000.
4. J. Bormans, J. Gelissen, and A. Perkis. MPEG-21: The 21st century multimedia framework. *IEEE Signal Processing Magazine*, 20(2), 2003.
5. M. Bosi and R.E. Goldberg. *Introduction to Digital Audio Coding and Standards*. Kluwer, 2003.
6. J.C. Brown. Determination of meter of musical scores by autocorrelation. *Journal of the Acoustical Society of America*, 94(4), 1993.
7. I. Burnett, R. Van der Walle, K. Hill, J. Bormans, and F. Pereira. MPEG-21: goals and achievements. *IEEE Multimedia*, 10(4), 2003.
8. M.J. Carey, E.S. Parris, and H. Lloyd-Thomas. A comparison of features for speech-music discrimination. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*, pages 149–152, 1999.
9. J.C. Catford. *Theoretical Acoustics*. Oxford University Press, 2002.
10. P. Cummiskey. Adaptive quantization in differential PCM coding of speech. *Bell Systems Technical Journal*, 7:1105, 1973.
11. T.F.W. Embleton. Tutorial on sound propagation outdoors. *Journal of the Acoustical Society of America*, 100(1), 1996.
12. H. Fletcher. Auditory patterns. *Review of Modern Physics*, pages 47–65, 1940.
13. A. Ghias, J. Logan, D. Chamberlin, and B.C. Smith. Query by humming: musical information retrieval in audio database. In *Proceedings of the ACM Conference on Multimedia*, pages 231–236, 1995.
14. A. Hanjalic and L.-Q. Xu. Affective video content representation and modeling. *IEEE Transactions on Multimedia*, 7(1):143–154, 2005.
15. X. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice-Hall, 2001.
16. L.E. Kinsler, A.R. Frey, A.B. Coppens, and J.V. Sanders. *Fundamentals of Acoustics*. John Wiley and Sons, New York, 2000.
17. P. Ladefoged. *Vowels and consonants*. Blackwell Publishing, 2001.

18. C.M. Lee and S.S. Narayanan. Toward detecting emotions in spoken dialogs. *IEEE Transactions on Multimedia*, 13(2):293–303, 2005.
19. L. Lu, H. Jiang, and H.J. Zhang. A robust audio classification and segmentation method. In *Proceedings of the ACM Conference on Multimedia*, pages 203–211, 2001.
20. Y.-F. Ma, X.-S. Hua, L. Lu, and H.-J. Zhang. A generic framework for user attention model and its application in video summarization. *IEEE Transactions on Multimedia*, 7(5):907–919, 2005.
21. B.S. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7*. John Wiley and Sons, Chichester, UK, 2002.
22. S.K. Mitra. *Digital Signal Processing - A Computer Based Approach*. McGraw-Hill, 1998.
23. B.C.J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, 1997.
24. P.M. Morse and K. Ingard. *Theoretical Acoustics*. McGraw-Hill, 1968.
25. P. Noll. Wideband speech and audio coding. *IEEE Communications Magazine*, (11):34–44, november 1993.
26. P. Noll. MPEG digital audio coding. *IEEE Signal Processing Magazine*, 14(5):59–81, 1997.
27. B.M. Oliver, J. Pierce, and C.E. Shannon. The philosophy of PCM. *Proceedings of IEEE*, 36:1324–1331, 1948.
28. A.V. Oppenheim and R.W. Schaffer. *Discrete-Time Signal Processing*. Prentice-Hall, 1989.
29. T. Painter and A. Spanias. Perceptual coding of digital audio. *Proceedings of IEEE*, 88(4):451–513, 2000.
30. J.O. Pickles. *An Introduction to the Physiology of Hearing*. Academic Press, 1988.
31. L. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 25(1):24–33, 1977.
32. L.R. Rabiner and M.R. Sambur. Algorithm for determining the endpoints of isolated utterances. *Journal of the Acoustical Society of America*, 56(S1), 1974.
33. L.R. Rabiner and R.W. Schaffer, editors. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.
34. E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*, pages 1331–1334, 1997.
35. A. Spanias. Speech coding: a tutorial review. *Proceedings of IEEE*, 82(10):1541–1582, 1994.
36. S. Sukittanon and L.E. Atlas. Modulation frequency features for audio fingerprinting. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*, pages 1773–1776, 2002.
37. E. Wold, T. Blum, D. Keislar, and J. Wheaton. Content-based classification, search and retrieval of audio. *IEEE Multimedia*, 3(3), 1996.



## Image and Video Acquisition, Representation and Storage

### What the reader should know to understand this chapter

- Elementary notions of optics and physics.
- Basic notions of mathematics.

### What the reader should know after reading this chapter

- Human eye physiology.
- Image and video acquisition devices.
- Image and video representation.
- Image and video formats and standards.
- Color representation.

### 3.1 Introduction

The eye is the organ that allows our brain to acquire the visual information around us. One of the most challenging tasks in the science consists in developing a machine that can *see*, that is it can acquire, integrate and interpret the visual information embedded in still images and videos. This is the topic of scientific domain called *image processing*. The topic of image processing is so large it cannot be described in a single chapter. Therefore for comprehensive surveys of this topic, the reader can refer to [10][23][27].

The aim of this chapter is to provide an introduction to the image and video acquisition, representation and storage. Image representation is the first step towards the realization of an image processing system (IPS) and a video processing system (VPS). A crucial aspect in the realization of an IPS and a VPS is the memory occupation. Therefore, we will pay special attention to image and video storage, describing the main formats.

The chapter is organized as follows: Sections 3.2 and 3.3 present, respectively, human eye physiology and the image acquisition devices; Section 3.4

discusses the color representation; Section 3.5 presents the main image formats paying special attention to JPEG; Sections 3.6 and 3.7 review video principles and the MPEG standard; in Section 3.8 some conclusions are drawn; finally, some problems are proposed at the end of the chapter.

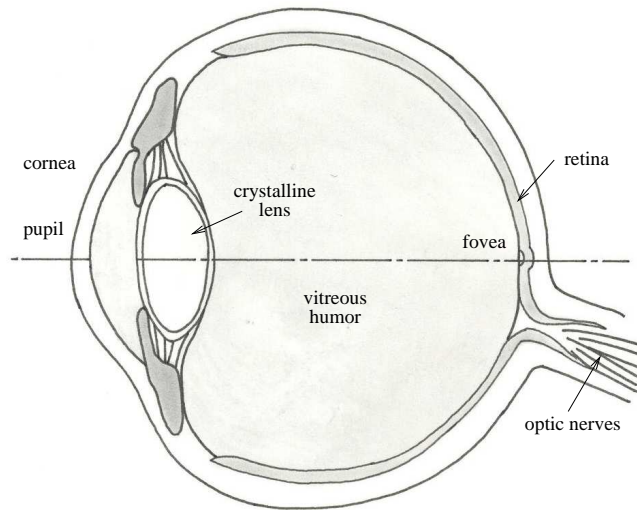
## 3.2 Human Eye Physiology

Electromagnetic radiation enters the human visual system through eyes and is incident upon the cells of the retina. Although human eyes can detect still images, they are mainly motion detectors. The eyes can identify static objects and establish spatial relationships among the different objects in a scene. Basic eye activity depends on comparing stimuli from neighboring cells. When we observe a static scene, our eyes perform small repetitive movements called *saccadic* that move edges past receptors. The *perceptual recognition* of human vision [30] takes place in the brain. The objects in a scene are recognized in the brain by means of their edges. The information about the object is embedded along these edges. The recognition process, i.e. the *perceptual recognition*, is a result of learning that is performed in the neural organization of the brain.

### 3.2.1 Structure of the Human Eye

The human eye, whose structure is shown in Figure 3.1, is the organ that gives us the sense of sight. Light reflected from an object enters the eye through the *cornea*, which is the clear dome at the front of the eye. Then the light enters through the *pupil*, the circular opening in the center of *iris*. The light passes through the *crystalline lens*, which is located immediately behind the iris and the pupil. Initially, the light waves are converged first by the cornea and then by the crystalline lens to a nodal point located immediately behind the back surface of the crystalline lens. At this stage of vision process, the image is reversed (turned backwards) and inverted (turned upside-down). The light passes through the *vitreous humor*, the clear gelatin that forms 80% of the overall volume of the eye. Finally, the light is focused on the *retina* which is located behind the vitreous humor. We can consider the eye a type of camera, as shown in Figure 3.2. In this metaphor the retina plays the role of the film, recording the light photons that interact with the retina.

The transport of the visual signal from the retina of the eye to the brain is performed through 1.5 million neurons by means of optic nerves. The human retina contains a big number of photoreceptors organized in a hexagonal array. The *retinal array* has three kinds of color sensors (or *cones*) in the central part of the retina (*fovea centralis*). The cone density is high in the fovea centralis and is low near the peripheral part of the fovea. In the retinal array there are three different kinds of cones, i.e. red, green and blue sensitive cones. These cones are responsible of color vision. The three cone classes have



**Fig. 3.1.** The human eye (picture by Matthias Dolder).

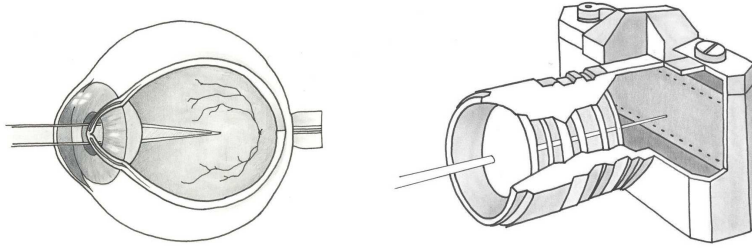
different photosensitive pigments. The three pigments have maximum absorptions at a wavelength of  $\sim 4,300$ ,  $5,300$  and  $5,600$  Angstrom (one Angstrom is equal to  $10^{-10}\text{m}$ ) which correspond, respectively, at *violet*, *blue-green* and *yellow-green*.

The space between the cones is filled by *rods* which are responsible for gray vision. The number of rods is larger than the number of cones.

Rods are sensitive to low levels of illuminations and are responsible for the human capability of seeing in dim light (*scotopic light*). The cones work at high illumination levels when many photons are available and the resolution is maximized at the cost of reduced sensitivity.

The optic nerve in human visual systems enters the eyeball and is put in connection with rods and cones. It starts as axon benches from the ganglion cells on one side of the retina. On the other side of the retina rods and cones are connected to the ganglion cells by means of bipolar cells. Besides, there are also horizontal nerve cells making lateral connections. The horizontal cells fuse signals from neighboring receptors in the retina forming a receptive field of opposing responses in the center and the periphery. Therefore a uniform illumination produces no stimulus. When the illumination is not uniform, a stimulus is produced. Some receptive fields use color differences. Therefore the color differences, in a similar way the one of illumination, produces stimuli, too.

In the human retina, the number of cones can vary from six to seven millions, whereas the number of rods ranges from 110 to 130 millions of rods. Transmission of the optical signals from rods and cones is performed



**Fig. 3.2.** The human eye can be viewed as a type of camera (pictures by Matthias Dolder).

by means of fibers in the optic nerve. The optic nerve crosses at the *optic chiasma*. In the chiasma the signals are dispatched to the brain, in particular the signals coming from the right and the left side of the retinas of two eyes are dispatched, respectively, to the right and the left halves of the brain. Each half of the brain receives half an image, so the loss of an eye in a person does not mean full blindness. The extremities of the optical nerve reach the lateral geniculate bodies and dispatch the signals to the *visual cortex*. The visual cortex has the same topology of the retina and represents the first step in the human visual perception since at this stage the visual information is available. Visual regions in two brain hemispheres are connected in the *corpus callosum*, which joins the visual field halves.

### 3.3 Image Acquisition Devices

A digital image acquisition is formed by two components, that is a *digital camera* and a host computer where the images acquired by the digital camera are stored. In the following sections we briefly describe how a digital camera works.

#### 3.3.1 Digital Camera

Digital cameras generally use either *charge coupled devices (CCD)* or *complementary metal oxide semiconductor (CMOS)* sensors and they can be grouped based on which of them they use.

In the CCD camera there is a  $n \times m$  rectangular grid of photo diodes (*photosensors*). Each photosensor is sensitive to light *intensity*. The intensity (or luminous intensity) is a measure of the power emitted by a light source in a particular direction. For the sake of simplicity, we can represent each photosensor with a black box that converts light energy into a voltage. The CCD array produces a continuous electric signal.

The structure of the CMOS camera is similar to the CCD one; the only difference is that the photo diode is replaced by a CMOS sensor. In each CMOS sensor there is a number of transistors that are used for the electric signal amplification. Since several transistors are used, the light sensitivity is lower since some photons are incident on the transistors instead of the photosensors. CMOS sensors are noisier than CCD sensors, but they consume less power and are less expensive.

When there is bright sunlight the camera aperture<sup>1</sup> does not have to be large since the camera does not require much light. On the other side, if the sunlight is not much, for instance when the sun is at sunset, the camera aperture has to be enlarged since the camera needs more light to form the image. The camera works like the human eye. The shutter speed<sup>2</sup> permits getting a measure of the exposure time of the camera to the light. In relation with the light requirement, the shutter opens and closes for an amount of time equal to the exposure time.

The *focal length* of a digital camera is given by the distance between the focal plane of the lens and the surface of the sensor grid. Focal length allows us to select the magnification degree which is requested to the digital camera.

The elementary unit of the digital image is the *pixel*, which is an abbreviation of *picture element*. A digital camera can capture images at different resolutions, i.e. using a different amount of pixels. A digital camera that works in low resolution usually represents an image using a matrix of  $320 \times 240$  (or  $352 \times 288$ ) pixels, whereas in medium resolution each image is generally represented by means of  $640 \times 480$  pixels. At high resolution the image is represented by  $1216 \times 912$  (or  $1600 \times 1200$ ) pixels. The *spatial resolution of an image* is the image size in pixels, for instance  $640 \times 480$ , which corresponds to the size of the CCD (or CMOS) grid.

Finally, we define two important parameters of the digital camera, i.e. the field of view and the sensor resolution. The *field of view* (or *FOV*) is the area of the scene that the digital camera can acquire. The FOV is fixed equal to the horizontal dimension of the inspection region that includes all the objects of interest. The *sensor resolution* (or *sensor size*) *SR* of a digital camera is given by:

$$SR = 2 \frac{FOV}{OR} \quad (3.1)$$

where *OR* stands for the minimum object resolution, i.e. the dimension of the smallest object that can be seen by the camera.

---

<sup>1</sup> The aperture controls the amount of light that reaches the camera sensor.

<sup>2</sup> In a camera, the shutter is a device that allows light to pass for a determined period of time, with the aim of exposing the CCD (or CMOS) sensor to the required amount of light to create a permanent image of view. Shutter speed is the time that the shutter is open.

## Color Acquisition

In this section we briefly review the process of color acquisition in the digital cameras. There are many methods for capturing colors. The typical approach uses red, green and blue (RGB) filters. The filters are spun in front of each sensor sequentially one after another, and separated images in three colors are stored at a fast rate. The digital camera acquires RGB components, given by the light intensity in the three wavelength bands, at each pixel location. Since each color component is represented by 8 bits it can assume 256 different values. Hence the overall amount of different colors that can be represented are  $256^3$  colors, i.e. each pixel can assume one among 16,777,216 colors.

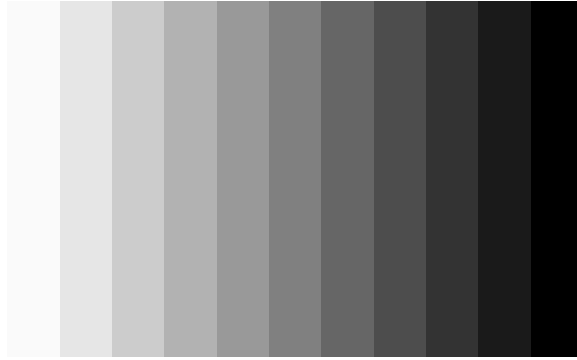
When we use the RGB filter strategy we make the implicit assumption that the colors in the image do not have to change passing from one filter to another one. This assumption in some cases cannot be fulfilled.

An alternative solution to RGB strategy is based on the *color interpolation* (or *demosaicing*). Demosaicing is a cheaper way of recording the RGB components of an image. According to this method only one type of filter over each photosensor is permanently placed. The sensor placements are usually carried out in accordance with a pattern. The most popular placement is the so-called *Bayer's pattern* [3]. In the Bayer's pattern each pixel is indicated by only one of the RGB components, i.e. the pixel is red, or green, or blue. It can make accurate guesses about the missing color component in each pixel location by means of demosaicing [24] [28].

High-quality cameras use three different sensors with RGB filters, i.e. one sensor for each RGB component. The light is directed to the sensors by means of a beam splitter. Each sensor responds to a narrow color wavelength band. Hence the camera acquires each of three colors for any pixel.

## Grayscale Image

A *grayscale* (or *graylevel*) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. Since a "gray" color is one in which the red, green and blue components all have equal RGB components, it is only necessary to specify a single intensity value for each pixel, unlike the three RGB components required to specify each pixel in a full color image. The grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. Grayscale images are entirely sufficient for many tasks (e.g. face recognition) and so there is no need to use more complicated and harder-to-process color images.



**Fig. 3.3.** A graylevel image

## 3.4 Color Representation

The elaboration of color images in image processing has been receiving more attention. This section introduces the basic principles underlying the human perception of color and reviews the main color models.

The light reflected from an object is absorbed by the cone cells and leads to the color perception. As we saw in Section 3.2, there are in the retina three different cone classes responsible for color perception. The human nervous system is sensitive to light intensity differences across different cones.

In this section we present the principles of human color perception and describe the main color models [18][29][30].

### 3.4.1 Human Color Perception

The electromagnetic radiation is perceptible by the human eye when its wavelength is between 4,000 and 7,700 Angstrom, i.e. between  $4 \cdot 10^{-7}$  and  $7.7 \cdot 10^{-7}$  m . The wavelengths of 4,000 and 7,700 correspond, respectively, to violet and red.

A color image can be represented as a function  $C(x, y, \lambda)$  where  $(x, y)$  individuates the point in the image and  $\lambda$  is the wavelength of the light reflected from the object. A *monochromatic image* is an image acquired in a fixed wavelength  $\lambda$ . The existence of three spectral perception functions  $V_R(\lambda)$ ,  $V_G(\lambda)$  and  $V_B(\lambda)$ , which correspond to three different types of cones, is the basis of color vision. The functions  $V_B(\lambda)$ ,  $V_G(\lambda)$  and  $V_R(\lambda)$  are maximal when the wavelengths are, respectively, 4,300, 5,300 and 5,600 Angstrom. These wavelengths do not correspond exactly to blue, green and red. Hence some researchers use the the nomenclature of *short-wavelength*, *medium-wavelength* and *long-wavelength* instead of the more popular R, G and B cones. The cones provide the human brain with color vision (*photopic vision*) and can distinguish small wavelength modifications. The eye sensitivity changes with the

wavelength and the maximum sensitivity corresponds to 5,070 Angstrom. An object in a scene, as perceived by an image acquisition device (e.g. a camera, a human eye), can be represented by a *radiance function*  $R(\lambda, x, y)$  where  $\lambda$  is the wavelength of a particular color at the point  $(x, y)$  of the scene. Weber formulated a relationship (*Weber's law*) between the physical stimuli from an object (e.g. the monitor luminance) and the subjective human perception. If we define  $W_L$  as the *just noticeable difference* (*JND*) in the brightness<sup>3</sup> required for distinguishing  $L$  and  $L + W_L$ , the following equation (*Weber's law*) holds:

$$\frac{W_L}{L} = k \quad (3.2)$$

where  $k$  is a constant, whose value is  $\sim 0.015$ .

Weber's law states that the larger the brightness  $L$  the larger the increase  $W_L$  required to perceive the difference between two objects of brightness  $L$  and  $L + W_L$ . On the other hand, distinguishing two objects of low brightness is much easier. If we have an object whose brightness is  $\frac{L}{10}$ , the increase in brightness  $w_l$  to distinguish another object will be smaller, that is will be one tenth of  $W_L$ .

More accurate investigations have proved that Weber's law does not always hold. In particular cases Weber's law has to be substituted by more precise formulae. For further informations, readers can refer to [5].

### Color Quantization

Actual computer monitors have generally  $256^3$  (i.e. 16,777,216) different colors (see Section 3.3). On the other hand, a human eye can usually distinguish only about 17,000 colors. Therefore, the usual color spaces of the actual computer monitors present a large redundancy if compared with the usual requirements of a human user. Removing the color redundancy generally improves the efficiency of color image processing algorithms. The color redundancy can be eliminated by mapping the usual color space onto a new space that has  $\sim 17,000$  colors (*color quantization*). In this way it can simulate the human color perception, preserving the image quality from a perceptive point of view.

The red color cones have minimum spectral sensitivity; green color cones have the maximum sensitivity, whereas blue color cones have an intermediate sensitivity. If we take into account the different sensitivity of three different color cones, the best policy consists of sampling in different ways the R, G, and B axes. Therefore, R-axis, B-axis and G-axis have, respectively, 24, 26 and 28 quantization levels. If we use these quantization levels, the overall amount of available colors is 17,472 which is approximately the same number of the perceived colors by the human eye. All the colors perceived can be seen as a linear combination of the *basic colors* (or *primaries*), that is red, green and blue. A human eye can distinguish two different colors only if there is a *JND*

---

<sup>3</sup> Brightness measures the color intensity (see Section 3.4.2).



(see Section 3.4.1) from each other. The JND is generally not constant due to the nonlinearity of the human vision. Buchsbaum investigated the visual nonlinearity of the human eye and his results are supported by physiology. For further information, readers can refer to [5].

### 3.4.2 Color Models

Many *color models* (or *color spaces*) have been proposed, and in each model color stimuli are represented by points in a three-dimensional *color space*. No model clearly outperforms the others and the best choice depends on the application. Color models [7] can be grouped in:

- *Colorimetric models* which are based on the physical spectral reflectance. An example of a colorimetric model is the *CIE chromaticity diagram*.
- *Physiologically inspired models* which are based on neurophysiology. Examples of these models are the *CIE XYZ* and *RGB models*.
- *Psychological models* which are based on how colors are perceived by the humans [13] [14]. An example of a psychological model is *HSB*.

Color models can be grouped [7] also in an alternative way:

- *Hardware-oriented color models*. These models are designed taking into account the properties of the devices (e.g. computer and TV monitors, printers) used to reproduce colors. Examples of hardware-oriented models are *RGB*, *CMY*, *YIQ* and *YUV*.
- *User-oriented color models*. These models are based on human perception of colors. Human color feel is based on *hue*, *saturation* and *brightness* perceptions. Hue indicates the wavelength of the perceived color. Saturation (or *chroma*) measures the quantity of white present in a color. Highly saturated colors (or *pure colors*) do not have any white component. Brightness (or *value*, or *intensity*, or *lightness*) measures the color intensity. Examples of user-oriented color models are *HLS*, *HCV*, *HSV* and *HSB*.

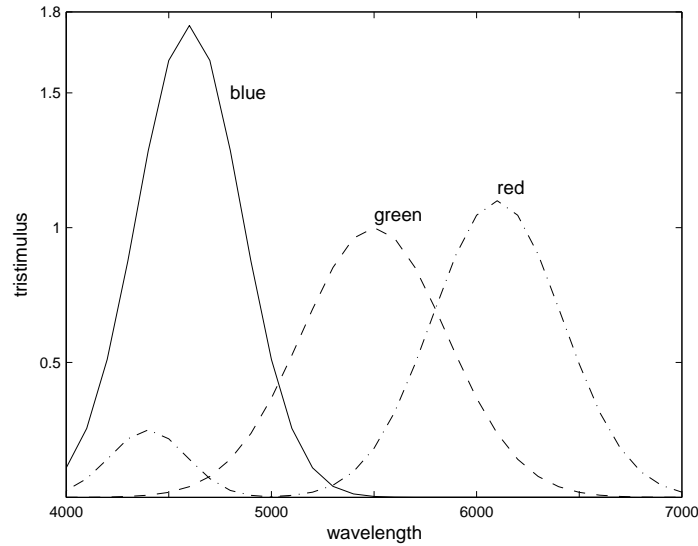
A review of the main color spaces is presented in the rest of the section. For more exhaustive presentation, readers can refer to [2][11][15][20][22][23][30].

### The Chromaticity Diagram

The research on color models has been carried out under the auspices of *Commission Internationale de l'Éclairage*<sup>4</sup> (*CIE*), an organization based in Paris. In the twentieth century CIE sponsored research into color perception which resulted in a class of mathematical models [30]. The common basis of these models consists in a collection of color-matching experiments, where an observer judges whether two parts of a visual stimulus (e.g. a figure) match in appearance, i.e. look identical or not. By varying the composition of the

---

<sup>4</sup> This is also called the *International Lighting Committee*.



**Fig. 3.4.** CIE 1931 standard observer color matching functions for virtual primaries. Blue, green and red correspond respectively to  $\mathbf{z}$ ,  $\mathbf{y}$  and  $\mathbf{x}$ .

light (i.e. an electromagnetic radiation visible to human eyes.) projected onto either part of the field of view, researchers can investigate properties of human color vision. It has been found that the light of almost any spectral composition (i.e. any color) can be *matched* by mixtures of three suitable chosen monochromatic primaries. A *monochromatic primary* is a light of a single wavelength. By repeating this kind of experiment with many different observers and averaging the results, and measuring the spectral composition and power of each of the light sources, the CIE has defined the so-called *standard observer color matching functions (SOCMF)*. Assuming that the human visual system behaves linearly, the CIE then went on to define the SOCMF in terms of the so-called *virtual primaries*. Virtual primaries are defined in such a way that SOCMF are all positive, which is desirable in practical applications. These primaries are called virtual since they cannot be physically obtained. The SOCMF for the virtual primaries are shown in figure 3.4. The SOCMF are usually called *CIE 1931 standard observer color matching functions*. The functions are generally indicated with  $\bar{x}$ ,  $\bar{y}$ ,  $\bar{z}$ . These functions are chosen such that  $\bar{y}$  is proportional to the human photopic luminosity function, which is an experimentally determined measure of the perceived brightness of monochromatic light of different wavelengths. These functions represent the basis of the research in color science, even though there have been many revision since 1931 [30]. If we know the spectral composition of a stimulus  $E(\lambda)$ , we can now determine its *chromaticity coordinates* as follows.

- First, we compute the *tristimulus values*  $X, Y, Z$

$$X = \int E(\lambda)\bar{x}(\lambda)d\lambda \quad (3.3)$$

$$Y = \int E(\lambda)\bar{y}(\lambda)d\lambda \quad (3.4)$$

$$Z = \int E(\lambda)\bar{z}(\lambda)d\lambda \quad (3.5)$$

- Then, we compute the *chromaticity coordinates*

$$x = \frac{X}{X + Y + Z} \quad (3.6)$$

$$y = \frac{Y}{X + Y + Z} \quad (3.7)$$

$$z = \frac{Z}{X + Y + Z} \quad (3.8)$$

If we add Equations (3.6), (3.7) and (3.8) we obtain:

$$x + y + z = 1 \quad (3.9)$$

Since  $z = 1 - (x + y)$ ,  $x$  and  $y$  are adequate to specify the chromaticity of a color. Therefore the chromaticity coordinates  $x, y$  are plotted forming the so-called *chromaticity diagram*. The chromaticity diagram has several properties. It represents every physically realizable color as a point. It has a white point at its center, with more saturated color radiating outwards from white. When superimposing light coming from two different sources, the resulting color perceived lies on a straight line between the points representing the component lights in the diagram. Moreover, we can represent the range of all colors that can be produced, the so-called *color gamut*, by means of three primaries as the triangular area of the chromaticity diagram whose vertices have coordinates defined by the chromaticities of the primaries. Now we pass to describe the main color models.

### RGB Color Model

The *RGB Color Model* is the most commonly used hardware-oriented color model. Color images in monitors and video cameras are represented in RGB (which is an acronym of Red Green Blue) space and they are usually called *RGB images*. Colors in RGB models are obtained as a linear combination of the primary colors red, green and blue. In the RGB model, RGB coordinates range from 0 to 1. They are connected with the tristimulus values  $X, Y, Z$  by means of the following equations:

$$X = 0.490R + 0.310G + 0.200B$$

$$Y = 0.177R + 0.831G + 0.010B$$

$$Z = 0.000R + 0.010G + 0.990B$$

In the RGB model, white and black are represented, respectively, by the triples  $(0,0,0)$  and  $(1,1,1)$ . red, green and blue are represented, respectively, by  $(1,0,0)$ ,  $(0,1,0)$  and  $(0,0,1)$ . Cyan, yellow and magenta, which are secondary colors obtained respectively by the superposition of green and blue, red and green, red and blue, are represented by the triples  $(0,1,1)$ ,  $(1,1,0)$  and  $(1,0,1)$ .

### CMY Model

CMY color model takes its name from the colors Cyan, Yellow, Magenta. Although these colors are secondary, cyan, magenta and yellow are the primary colors of pigments. Cyan, yellow and magenta are called *subtractive primaries* because these colors are obtained by subtracting light from white. The CMY model finds application in color printers. The transformations that allow us to pass from RGB to CMY model can be obtained transforming RGB values into XYZ and then from XYZ coordinates into CMY. An approximate transformation, inaccurate in some cases, that allows us to pass from RGB to CMY model is the following:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 - C \\ 1 - M \\ 1 - Y \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (3.10)$$

where  $R, G, B \in [0, 1]$ .

### YIQ and YUV Models

The YIQ model represents the grayscale information by means of the *luminance*  $Y$ . Whereas *hue*  $I$  and *saturation*  $Q$  express the color information and are often called *chrominance components*. YIQ coordinates can be obtained from the RGB model using the following transformation:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ I &= 0.596R - 0.274G - 0.322B \\ Q &= 0.211R - 0.523G + 0.312B \end{aligned}$$

The YIQ model is used in NTSC (*National Television Standard Committee*), which is the television standard in the USA, and for this reason is also called the *NTSC color space*.

The YUV model is similar to YIQ. The grayscale is represented by means of the *luminance* whereas the chrominance components are  $U$  (or  $C_b$ ) and  $V$  (or  $C_r$ ).  $C_b$  and (or  $C_r$ ) are respectively called *blue difference component* and *red difference component*. YUV coordinates can be obtained from the RGB model using the transformation (also called *television law*):

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= 0.493(B - Y) \\ V &= 0.877(R - Y) \end{aligned} \quad (3.11)$$

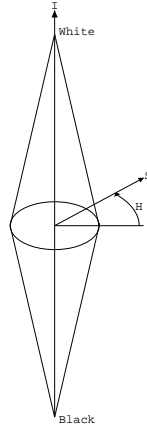
Equations (3.11) fully justify the names of the chrominance components. The YUV model is used in *PAL*, which is the television standard in Europe (with the exception of France where the standard is *SECAM*).

### User-Oriented Color Models

Although RGB, CMY and YIQ models are useful for color representation, they are not similar to human perception. A drawback of the RGB model is the lack of uniformity. A *uniform color space* is a space where the Euclidean distance between two color points corresponds to the perceptual difference between two corresponding colors in the human vision system. In other words, in a nonuniform color space, two couples of color points with the same distance do not show the same degree of perceptual difference. In imaging applications, it is very popular the use of *perceptually uniform color spaces*. Hence the nonuniform RGB space has to be transformed into any perceptually uniform space. Before we describe these spaces, it is necessary to remark on some facts described in the following.

Color is an attribute of human visual perception and can be described by color names such as green, blue and so on. Hue is another attribute of human perception and can be described by *primary hues* (red, green, blue, purple and yellow) or by a combination of them. Although black, white and gray are colors, they are not classified by CIE as hues. Therefore perceived colors can be divided into two families: *achromatic colors* and *chromatic colors*. Achromatic colors that are not hues (i.e. Black, White and gray); chromatic colors that are hues. Hue, described already as a color property of light, can be also thought as a property of the surface reflecting or transmitting the light. For instance, a blue glass reflects blue hue. Hence hue is also an attribute of the human perception and is the chromatic component of our perception. It can be classified as *weak hue* or *strong hue*. The colorfulness of a color is expressed by the *saturation*. For instance, the color from a single monochromatic source of light, which yields the color of a unique wavelength, is highly saturated, whereas the colors that have hues of different wavelengths have small chroma and less saturation. For example, gray colors do not have hues and their saturation is null (*unsaturated colors*). Hence the saturation can be seen as a measure of colorfulness (or the whiteness) of the color in the human perception. The *lightness* ( $L$ ), also called *intensity* ( $I$ ) or *value* ( $V$ ), measures the color *brightness*. It provides a measure of how much light is reflected from the colored object or how much light is emitted from a region. The lightness is proportional to the electromagnetic energy emitted by the object. Finally, the luminosity helps the human eye in color perception. For instance, a colored object in the darkness does not appear colorful at all. That being stated, we pass to describe the color models based on human perception of colors (also called *user oriented color models*).

The first user-oriented color model was proposed by Munsell [9][16][21] about 90 years ago. His model, called *the Munsell color space*, was designed

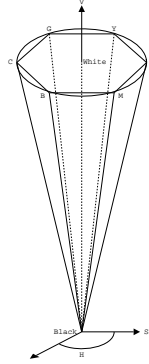


**Fig. 3.5.** HIS color space.  $I$ ,  $S$  and  $H$  indicate, respectively, intensity, saturation and hue.

for artists and based on subjective human assessments rather than on objective perceptual measures (e.g. measurements of hue, saturation and brightness). The Munsell color model uses a cylindrical coordinate scheme and is too cumbersome to be used in imaging application. Therefore, several approximations of the Munsell color model have been developed. They separate luminance from the other components, supporting in this way an intuitive notion of color. Among these models, the most popular are *HIS* (*hue, intensity and saturation*), *HCV* (*hue, chroma and value*), *HSV* (*hue, saturation, value*) and *HSB* (*hue, saturation, brightness*). These models are closely related. Color coordinates can be derived from RGB and XYZ models by means of generally nonlinear equations. These models are very popular in image processing. For the sake of space, we will only describe *HIS*, *HSB*, *HSV*.

The *HIS* model, where HIS stands for *hue, intensity and saturation*, can be represented by means of a double cone (see Figure 3.5). Gray is in the middle of the axis whereas white and black are located, respectively, in the top and in the bottom cone vertex. Hue and saturation are represented, respectively, by the angle around the vertical axis and the distance from the central axis. Most saturated colors are located close to the maximum circle. Primary and secondary colors are located on the maximum circle equally spaced at 60 degrees: red, yellow, green, cyan, blue, magenta (listed counterclockwise).

The *HSV* model, where HSV stands for *hue, saturation and value*, is strictly related to HCV, HLS and HSI and it can be represented by a cone (see Figure 3.6). Similarly to the HIS model, the cone axis represents the line of gray. HSV coordinates can be obtained from the RGB model using different transformation. The simplest transformation is the following:



**Fig. 3.6.** HSV color space.  $C, G, Y, R, M, B$  indicate respectively cyan, green yellow, red, magenta and blue. Black and white have, respectively,  $V = 0$  and  $V = 1$ .

$$V = \frac{R + G + B}{3}$$

$$S = 1 - \frac{\min(R, G, B)}{V} \quad (3.12)$$

$$H = \tan \left[ \frac{3(G - B)}{(R - G) + (R - B)} \right]. \quad (3.13)$$

Note that  $H$  is undefined when  $S = 0$ .

The most popular HSV transformation is the following. Firstly, RGB values are normalized by defining:

$$r = \frac{R}{R + G + B}; \quad g = \frac{G}{R + G + B}; \quad b = \frac{B}{R + G + B}. \quad (3.14)$$

Then  $H, S, V$  can be computed using:

$$V = \max(r, g, b) \quad (3.15)$$

$$S = \begin{cases} 0 & \text{if } V = 0 \\ V - \frac{\min(r, g, b)}{V} & \text{if } V > 0. \end{cases} \quad (3.16)$$

$$H = \begin{cases} 0 & \text{if } S = 0 \\ \frac{60 * (g - b)}{S * V} & \text{if } V = r \\ 60 * \left[ 2 + \frac{b - r}{S * V} \right] & \text{if } V = g \\ 60 * \left[ 4 + \frac{r - g}{S * V} \right] & \text{if } V = b \end{cases} \quad (3.17)$$

$$H = H + 360 \quad \text{if } H < 0. \quad (3.18)$$

The *HSB* model, where HSB stands for *hue*, *saturation* and *brightness*, is inspired by Hurvich and Jameson's opponent colors theory [14] which is based on the observation that opponent hues (yellow and blue, green and red) erase each other when superimposed. Hurvich and Jameson computed the relative

quantity (*chromatic response functions*) of each of four basic hues present in each stimulus at a given wavelength. Besides, Hurvich and Jameson fixed the relative quantity of each of the four basic hues in each stimulus at a given wavelength which represents the perceived brightness of a visual stimulus at a given spectral composition. Hue and saturation coefficients function were derived by means of chromatic and achromatic response functions. *Hue coefficient functions* represent hue by means of the ratio between each chromatic response and the total of chromatic responses at each wavelength. *Saturation coefficient functions* represent saturation by means of the ratio between the total of chromatic responses and the achromatic response at each wavelength. HSB is polar coordinate model and reproduces with some accuracy many psychophysical phenomena. HSB coordinates  $rg$ ,  $by$  and  $wb$  can be obtained from RGB model by means of the following equations:

$$\begin{aligned}rg &= R - G \\by &= 2^B - R - G \\wb &= R + G + B.\end{aligned}\tag{3.19}$$

Finally, the intensity axis  $wb$  can be sampled more roughly than  $rg$  and  $by$  without a human observer noticing any perceptible differences.

### 3.5 Image Formats

Storage and retrieval of images is performed by means of files. They are organized on the basis of particular standards called *image file format standards*. Storing an image requires a lot of memory. For instance, a grayscale image of  $1024 \times 1024$  requires  $1024 \times 1024$  bytes i.e. 1 MByte. Therefore each image is stored in compressed form. Image file formats can be divided into two families: *no lossy image file formats* and *lossy image file formats*. In the no lossy image file formats the compression stage does not imply an information loss. Hence after the decompression we obtain the original file before the compression. Vice versa, in the lossy formats the compression stage implies an information loss.

#### 3.5.1 Image File Format Standards

In this subsection we will provide a concise description of the most popular image file formats with the exception of JPEG. This standard will be presented in Section 3.5.2.



**Tagged Image File Format (TIFF)**

This format, whose file extension is *.tif* or *.tiff*, can be used to efficiently manage very different types of images such as, for instance, bitmaps and compressed color images. TIFF is generally a *no-lossy compression* format<sup>5</sup>.

**Portable Network Graphics (PNG)**

PNG, whose file extension is *.png*, is a format that provides lossless storage of raster images. PNG offers the main TIFF functionalities.

**Graphics Interchange Format (GIF)**

GIF supports 8-bit color images and is generally used in application programs (e.g. word processors) in the Windows environment.

**Postscript**

This format, developed in the UNIX environment, is used for printing. In this format gray-level images are represented by decimal or hexadecimal numerals written in the ASCII format.

**Portable Image File Formats**

Portable image file formats are very popular image file formats which include *portable bitmap*, *portable graymap*, *portable pixmap* and *portable network map*, whose file extensions are, respectively, *.pbm*, *.pgm*, *.ppm* and *.pnm*. Portable image file formats are convenient methods for the storage and retrieval of the images since they supports all kinds of images of increasing complexity, ranging from bitmaps to color images.

**PPM and PGM File Formats**

A PPM file is organized into two different parts, a header and the image data. The header contains a PPM identifier (*P3* and *P6*, respectively, for ASCII and binary formats), the width and the height of the image coded in ASCII and the maximum value of the color components of the pixels. The PGM format allows us to store only grayscale images. Its format is identical to PPM with the unique difference in the identifier of the header (*P2* and *P5*, respectively, for ASCII and binary formats).

---

<sup>5</sup> TIFF also provides lossy compression schemes, although they are less popular.

## PBM

PBM format allows us to store binary images as a series of ASCII 0 (white pixel) or 1 (black pixel). The PBM header is identical to the one of PPM format with the only difference of the header. The header contains a PBM identifier (*P1*), the width and the height of the image coded in ASCII .

### 3.5.2 JPEG Standard

*JPEG*, whose file extension is *.jpg*, is the acronym of *Joint Photographic Experts Group*. JPEG is the first international image compression standard for continuous-tone still images (e.g. photos). This standard is the result of joint efforts by the *International Telecommunication Union (ITU)*, *International Organization for Standardization (ISO)* and *International Electrotechnical Commission (IEC)* and is referred as *ISO/IEC IS 10918:1: Digital Compression and Coding of Continuous-tone Still Images*. JPEG is very important since the video standard MPEG is based on JPEG. For this reason, we pay particular attention to this standard. JPEG generally performs a *lossy compression*, i.e. the compression implies an information loss and the image after the decompression stage is not identical to the original image. JPEG has four modes (*sequential lossless mode*,<sup>6</sup> *sequential DCT-based mode*, *progressive DCT-based mode*, *hierarchical mode*) and several options. For the sake of space, we will describe only the JPEG basic coding algorithm (*baseline JPEG algorithm*) which is based on *Huffman coding* for *entropy encoding*.

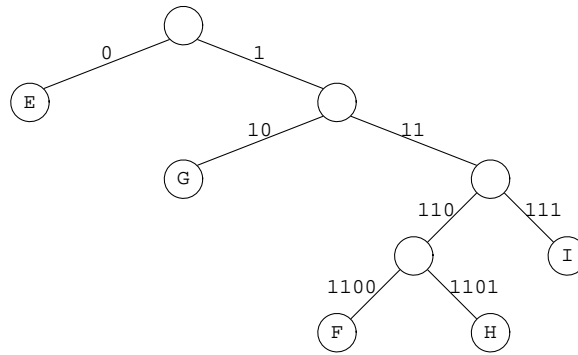
### Huffman Coding

*Huffman coding* [12] is a popular and effective method of lossless data compression. It is a form of *entropy coding*. In order to present Huffman coding, we consider the following example [6]. We have a data file formed by 50,000 characters of only five types, for instance *E*, *F*, *G*, *H*, *I*. Besides, we suppose that the frequency of the characters in the file is known (see Table 3.1). Our goal is to represent each character with a *binary code* (or *code*). If we use a *fixed length code* we need three bits for represent five characters, as shown in Table 3.1. Hence the overall amount of bits required for coding the file is

**Table 3.1.** Frequency of each character in the file.

|                 | <i>E</i> | <i>F</i> | <i>G</i> | <i>H</i> | <i>I</i> |
|-----------------|----------|----------|----------|----------|----------|
| Frequency       | 40%      | 5%       | 25%      | 10%      | 20%      |
| Fix Length Code | 000      | 001      | 010      | 011      | 100      |
| Huffman Code    | 0        | 1100     | 10       | 1101     | 111      |

<sup>6</sup> In this mode, JPEG produces a lossy compression.



**Fig. 3.7.** Huffman tree representation for the example of Table 3.1.

150,000 bits. Can a code be designed that requires less bits? The answer to this question is provided by Huffman coding [12]. David Huffman proposed his code, when he was a MIT graduate student, as a exam project for the information theory course. Huffman's basic idea was to represent each character by means of a binary string of variable size (*variable length code*). In this way a shorter bit code was associated with the character whose frequency was higher. Huffman coding for the five characters, shown in Table 3.1, requires an overall amount of bits  $B_H$  equal to:

$$B_H = 50,000 * (0.40 * 1 + 0.05 * 4 + 0.25 * 2 + 0.10 * 4 + 0.20 * 3) = 105,000.$$

Huffman coding, compared with fix length code, lets us save 45,000 bits, i.e. 30% of the overall required storage space. It is possible to show [6] that Huffman coding is optimal. The key to Huffman coding is *Huffman's algorithm* which makes an *extended binary tree* of *minimum weighted path length* from a list of weights. We now describe the algorithm. Firstly, we assume that with each symbol (or character) is associated a weight equal to the number of the symbol occurrences in the file. For instance, in our example, with the symbols  $E$  and  $I$  are associated, respectively, 40 and 20. Huffman's algorithm uses a *bottom-up strategy* and assumes that we are making a unique tree starting from a group of trees (*forest*). In the first stage, each tree is composed of a single node with the associated symbol and weight. Trees are gathered by choosing two trees and creating a new tree from the fusion of the two original trees. Hence the forest cardinality decreases by one unity at each algorithm stage. When the forest is composed by a unique tree, Huffman's algorithm stops. Huffman's algorithm is composed of the following steps:

1. Start with a forest of trees. Each tree is composed of a unique node, with an associated character and weight. The weight is equal to the occurrences of the character.
2. Pick two trees ( $T_1$  and  $T_2$ ) with the smallest weights of roots. Create a new tree ( $T_n$ ), whose left and right subtrees are respectively  $T_1$  and  $T_2$ ,

which has a root whose weight  $w(T_n)$  is equal to:

$$w(T_n) = w(T_1) + w(T_2)$$

where  $w(T_1)$  and  $w(T_2)$  are, respectively, the weights of  $T_1$  and  $T_2$ .

3. If the forest cardinality is more than one go to step 2; otherwise return the single tree left.

It is possible to show that the single tree returned by Huffman's algorithm is an *optimal encoding tree* [6]. The labeling of the edges of the optimal encoding tree is arbitrary. A popular strategy consists in assigning a value of 0 to an edge of any left child and a value of 1 to an edge of any right child (or vice versa). By concatenating the labels of the edges we obtain the Huffman coding. The labeled optimal encoding tree, produced by Huffman's algorithm, in the example of five characters is shown in Figure 3.7. Finally, we conclude with the remark that Huffman's algorithm is an example of *greedy algorithm* [6]. It is greedy since the nodes with the smallest weights are picked at each step and this *local optimal decision* results in a global optimal encoding tree.

### Baseline JPEG Algorithm

After the description of Huffman's coding we return to JPEG and describe its baseline algorithm. The baseline JPEG algorithm is formed by the following steps:

1. *Color space transformation*: Firstly, the image is converted from RGB space into a  $Y C_b C_r$  space, similar to YIQ and YUV color spaces used in NTSC and PAL systems. As we have seen previously,  $Y$  is the luminance component whereas  $C_b$  and  $C_r$  components together represents the image *chrominance*. A matrix for each single component is built. Each matrix is formed by elements whose range is from 0 to 255.
2. *Downsampling*: The chrominance components are downsized. Each  $C_b$  and  $C_r$  matrices are reduced by a factor of two in horizontal and vertical directions.<sup>7</sup> This is performed by averaging on squares formed by four pixels. For instance, if each matrix, before downsampling, had  $640 \times 480$  pixels, after downsampling  $Y$  matrix has  $640 \times 480$  pixels, whereas  $C_b$  and  $C_r$  matrices have  $320 \times 240$  pixels. Downsampling is a *lossy data compression* but it is not practically noticed by the human eye since it is more sensible to the luminance signal than the chrominance ones. The element values are then centered around zero by subtracting 128 from each one of them. Finally each matrix is divided in blocks of  $8 \times 8$  pixels.
3. *Discrete cosine transform*: The  $8 \times 8$  blocks of each component ( $Y$ ,  $C_b$ ,  $C_r$ ) are converted to the frequency space using a two-dimensional *discrete cosine transform (DCT)* (see the Appendix). DCT output is a  $8 \times 8$  matrix

---

<sup>7</sup> JPEG offers the possibility of reducing by a factor of 2 only in the horizontal direction.

|     |    |    |    |   |   |   |   |
|-----|----|----|----|---|---|---|---|
| 145 | 76 | 43 | 16 | 5 | 2 | 1 | 0 |
| 97  | 79 | 39 | 12 | 7 | 1 | 0 | 0 |
| 48  | 35 | 24 | 7  | 6 | 4 | 0 | 0 |
| 16  | 11 | 8  | 5  | 2 | 1 | 0 | 0 |
| 7   | 5  | 3  | 0  | 0 | 0 | 0 | 0 |
| 4   | 3  | 1  | 1  | 0 | 0 | 0 | 0 |
| 2   | 1  | 0  | 0  | 0 | 0 | 0 | 0 |
| 0   | 0  | 0  | 0  | 0 | 0 | 0 | 0 |

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 1  | 1  | 2  | 4  | 8  | 16 | 32 | 64 |
| 1  | 1  | 2  | 4  | 8  | 16 | 32 | 64 |
| 2  | 2  | 2  | 4  | 8  | 16 | 32 | 64 |
| 4  | 4  | 4  | 4  | 8  | 16 | 32 | 64 |
| 8  | 8  | 8  | 8  | 8  | 16 | 32 | 64 |
| 16 | 16 | 16 | 16 | 16 | 16 | 32 | 64 |
| 32 | 32 | 32 | 32 | 32 | 32 | 32 | 64 |
| 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

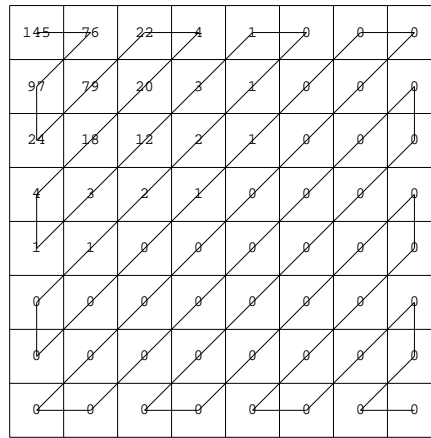
|     |    |    |   |   |   |   |   |
|-----|----|----|---|---|---|---|---|
| 145 | 76 | 22 | 4 | 1 | 0 | 0 | 0 |
| 97  | 79 | 20 | 3 | 1 | 0 | 0 | 0 |
| 24  | 18 | 12 | 2 | 1 | 0 | 0 | 0 |
| 4   | 3  | 2  | 1 | 0 | 0 | 0 | 0 |
| 1   | 1  | 0  | 0 | 0 | 0 | 0 | 0 |
| 0   | 0  | 0  | 0 | 0 | 0 | 0 | 0 |
| 0   | 0  | 0  | 0 | 0 | 0 | 0 | 0 |
| 0   | 0  | 0  | 0 | 0 | 0 | 0 | 0 |

**Fig. 3.8.** Quantization process in JPEG. (a) DCT matrix before the quantization; (b) quantization matrix; (c) DCT matrix after the quantization

of *DCT coefficients*. Theoretically, DCT is no lossy, but practically there is a small information loss due the approximation errors.

4. *Quantization*: The human eye can detect a small difference in brightness, but is not able to discriminate the exact magnitude of a high-frequency brightness variation. This physiological fact is used in JPEG to reduce the amount of information in the high frequencies. This is performed in the Quantization step, where less important DCT coefficients, generally the ones related to high frequencies, are deleted. This lossy transformation is performed by dividing each DCT coefficient by a weight taken from a table (*quantization table*). If all weights are 1, the transformation produces no effects, but if the weights increase quickly from the origin, the coefficients related to high frequency are downsized notably. An example of the quantization process is shown in Figure 3.8.
5. *Average value reduction*: In this step the value (0,0) (*average value*) of each block, which is given by the value at the top left corner, is reduced, by replacing it with the difference between actual average value and the average value of the previous block. This difference is generally small since the average values of the block does not differ each other notably. Hence replacing each average value with its difference with the average value of the previous block implies that most average values, after *average value reduction*, are very small. During average value reduction, the other DCT coefficients do not change.
6. *Linearization*: In this step the *linearization* of the block is performed . The block is linearized using a particular *zig-zag scheme*, shown in Figure 3.9. The zig-zag scheme produces a density of zero at the end of the block. In Figure 3.9 the zig-zag scheme produces a final sequence of zeros which is effectively coded using a unique value, i.e. the zero amount. At the end of the linearization process, the image is represented by a unique list of numbers.
7. *Huffman coding*: Finally, the list of number is coded by Huffman coding.

JPEG is very popular since its compression rate is generally not less than 20:1. The decoding of a JPEG image requires performing the above-described



**Fig. 3.9.** Linearization of the block. The order is from left to right.

algorithm backwards. The encode and the decode of a JPEG image generally require the same computational resources.

### 3.6 Video Principles

A property of the human eye is to hold for a few milliseconds the projected image of any object before it dissolves. If a sequence of image is projected at more than 25 images per second, human eyes cannot realize that they are looking at a sequence of discrete images. Video and movies use this principle to produce the sensation of moving images. To understand video, the best approach [26] is to consider the model of black-and-white television. To represent the bidimensional image, the camera makes a scanning, by means of a beam of electrons, fast from left to right and more slowly from up to down recording the light intensity on the screen. When the scanning is complete (*frame*), the electron beam restarts. The intensity, in function of time, is the transmitted signal and receivers repeat the scanning to reproduce the image. Although modern CCD videocameras make an integration instead of a scanning, some videocameras and CRT<sup>8</sup> monitors make a scanning. Hence, our description has still a certain degree of validity. The parameters of the scanning depends on the considered television standard. *NTSC* (*National Television Standard Committee*), the television standard in USA, has 525 scanning lines, the ratio between the horizontal and the vertical dimension is  $\frac{4}{3}$  and makes 30 frames per second, whereas, the European standards *PAL* (*Phase Alternative Line*) and *SECAM* (*SEquentiel Couleur Avec Memoire*)<sup>9</sup> have 625 scanning

<sup>8</sup> CRT stands for *cathode-ray tube*.

<sup>9</sup> Sequential Color with Memory.

lines, the same ratio of  $\frac{4}{3}$  between the horizontal and the vertical dimension and make 25 frames per second. The color television uses the same scanning model of the black-and-white television. In this case three synchronized electron beams are used, one beam for each of three primary colors (Red, Green and Blue). Then, in the three television systems (i.e. NTSC, PAL and SECAM) RGB signals are transformed into a *luminance* signal and into two *chrominance* signals. Each system uses different transformations to obtain chrominance signals. Since the human eye is more attuned to the luminance, the luminance has to be transmitted more accurately than the chrominance signals.

We have briefly described the analog television. We describe now digital video. Digital video is a sequence of frames, each of them is a digital image, whose basic element, as we have seen, is the pixel. In digital video color, each primary color (i.e. red, green and blue) is represented by eight bits. Hence more than sixteen millions of colors can be represented in the digital color videos. As we have seen at the beginning of this chapter, human eyes can distinguish only a smaller number of colors, i.e.  $\sim 17,000$  colors.

In order to produce a uniform movement, digital video has to display at least 25 frames per second. In digital video, the rate between the horizontal and the vertical dimension is  $\frac{4}{3}$ , whereas the digital screen usually has  $640 \times 480$  (or  $800 \times 600$ ) pixels.

*High-definition television* standards have different parameters, the digital screen has  $1280 \times 720$  pixels and the rate between the horizontal and the vertical dimension is  $\frac{16}{9}$ . For sake of precision, we have to underline that the European standard *digital video broadcasting (DVB)* also permit  $\frac{16}{9}$  as rate between the horizontal and the vertical dimension.

In the next section we will describe the main standard for video compression, i.e. *MPEG*.

### 3.7 MPEG Standard

Video requires a huge quantity of memory for the storage. For instance, a TV movie without compression, displayed on a screen of  $640 \times 480$  pixels with a length of two hours, requires about 200 GBytes. Hence, compression is a crucial topic for digital video. In this section, we briefly describe the MPEG standard, paying particular attention to MPEG-2. The *MPEG (Motion Picture Experts Group)* [19]. *MPEG-1* (International Standard 11172) was designed for a videorecorder at 1.2 Mbps. *MPEG-2* (International Standard 13118) was designed to compress video signals from 4 to 6 Mbps in order to be used in NTSC and PAL television systems. Both MPEG-1 and MPEG-2 use spatial and temporal redundances in the video. A *spatial redundancy* can be exploited coding separately each frame by means of JPEG. A further compression can be obtained observing that consecutive frames are often almost the same (*temporal redundancy*). The *digital video* system (*DV*), used in digital

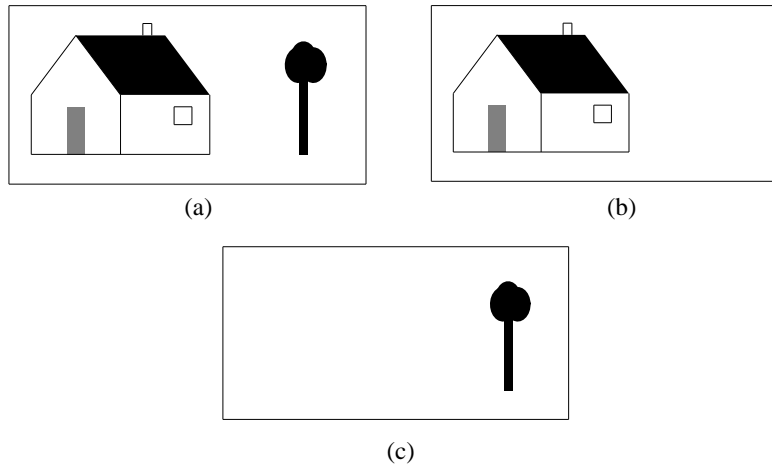
videocameras, codes each frame separately by means of JPEG. Since coding has to be performed in real time, coding each frame separately is faster. In the scenes where the videocamera and the landscape are fixed and only one or two objects move slowly, almost all pixels will be the same in two consecutive frames. Therefore, a good compression result can be obtained subtracting each frame from the preceding frame and performing JPEG compression on the difference. This is the strategy adopted by MPEG. Nevertheless, when the videocamera performs a zoom, this strategy fails. Therefore a method of motion compensation is required. This is the strategy adopted by MPEG and is the main difference between JPEG and MPEG. MPEG-2 produces three different frame types, which have to be elaborated by the display program. The frames are:

- *I-frame* (or *intra-frame*): still images coded by means of JPEG.
- *P-frame* (or *predictive frame*): the difference between the actual frame and its predecessor.
- *B-frame* (or *bidirectional frame*): differences between the actual frame and its predecessor and its successor.

I-frames are still images coded by means of JPEG. This implies that the luminance is used at full resolution, whereas the chrominance components are used at half resolution along both horizontal and vertical axes. I-frames have to be produced periodically for some reasons. Firstly, MPEG can be used for the television transmission which is characterized by the fact that the customers connect themselves with the television transmission when they want. If all the frames depend on the preceding one, anyone who has missed the first frame could never decode the succeeding frames. Besides, if a frame was received wrongly, it could not decode the succeeding frames.

P-frames code the differences between two consecutive frames. They are based on the idea of *macroblocks*, which cover  $16 \times 16$  pixels in luminance and  $8 \times 8$  pixels in the chrominance components. A macroblock is coded looking for in the preceding frame the same macroblock or a macroblock which differs only a little from it. An example of P frame is shown in Figure 3.10. The B-frames are similar to P-frames, with the difference that they code both the differences of the actual frame with the preceding and the succeeding frame. To code a B-frame, the decoder requires to maintain, at the same time, in the memory three frames: the preceding one, the actual one and the succeeding one. In order to make simpler the coding, the frames are ordered in a MPEG-flux on the basis of their dependence and not on the basis of the order according to which they are displayed. In the next section we will describe other MPEG standards.





**Fig. 3.10.** MPEG-2 standard. (a) and (b) are two consecutive I frames; (c) is the P-frame, which is obtained subtracting (b) from (a).

### 3.7.1 Further MPEG Standards

After the success of MPEG-2, further standards of the MPEG family have been developed. We briefly summarize below MPEG-4, MPEG-7 and MPEG-21.

#### MPEG-4 Standard

MPEG-4 supports the composition of audiovisual information and representation of media in multimedia environments. MPEG-4 provides a toolbox that has tools and algorithms for content-based interactivity, compression and access. In particular, the toolbox contains *content-based multimedia data access tools*, *content-based manipulation and bitstream editing*, *natural and synthetic data coding*, *improved temporal random access*, *improved coding efficiency and coding of multiple concurrent data streams*, *robustness to errors and content-based scalability* [17]. MPEG-4 describes audiovisual data in the form of objects. MPEG-4 objects are entities that combine a data structure (*object state*) with a set of methods (*object behavior*). A method is a computable procedure associated with an object that works on data structures. MPEG-4 provides a number of predefined classes organized in a hierarchical way. Classes are object templates (e.g. images, audio clips). The hierarchy identifies the relationships among classes, in particular the inheritance. For instance a part of an image inherits the properties of the whole image (e.g. gray-scale). The set of classes above described is called *MPEG-4 standard class library*.

The architecture of MPEG-4 uses a *terminal model* for transmitting audiovisual data. A *MPEG-4 terminal* assumes a twofold form, i.e. it can be

either a standalone application or part of a multimedia terminal. The former terminal (*encoder*) encodes and transmits audiovisual data through a communication network. The latter terminal (*decoder*) decodes and displays the audiovisual data. In the encoder terminal, audiovisual data are compressed, error protected and then transmitted under the form of binary streams. In the decoder terminal, the binary streams are corrected, whenever it is necessary, and decompressed. Then a *compositor* presents and renders the objects on the screen. The objects of a scene are memorized with the related information about their relationships. This information is used by the compositor to display the complete scene. MPEG-4 offers two different terminal kinds: *nonflexible* and *flexible*. Nonflexible terminals are based on a set of algorithms and profiles which are combined to offer a set of predefined classes which can be chosen by the user by means of switches. Flexible terminals permit the transmission of new classes defining, in this way, new templates for the transmitted audiovisual data.

Now we describe MPEG-4 representation. The video verification model of MPEG-4 provides a set of classes for the representation of the structure and content of an audiovisual sequence [8]. A video sequence is modelled in terms of a set *video sessions*. A video session is a collection of one or more *video objects*. Each video object has one or more *video object layers*. Video objects form an audiovisual scene and have properties (e.g. shape and texture). Each video object layer provides the temporal or spatial resolution of a video object. The layer is formed by an ordered sequence of snapshots (*video object planes*)(*VOPs*). Each video object plane is a video object at a given time. The VOP bounding box is divided into a number of macroblocks of  $16 \times 16$  pixels and are coded by means of JPEG. Binary or gray-scale shape information can be associated with video objects. Binary shape information identifies the pixels which belong to the video object. Binary shape information is expressed by a matrix which has the same size of the VOP bounding box. In a similar way, gray-scale shape information is also expressed by means of a matrix and represented, with a value from 0 to 255, the transparency degree of the pixels. gray-scale shape information is encoded by JPEG.

Motion estimation and compensation is made by splitting each VOP into macroblocks of  $16 \times 16$  pixels and by matching motion estimation. Each VOP can be coded in three different ways, that is *I-VOP* (or *intra VOP*), *P-VOP* (or *predicted VOP*) and *B-VOP* (or *bidirectional VOP*). I-VOPs are encoded in a complete independent way; P-VOPs are predicted from the preceding VOP. B-VOPs are interpolated from the preceding and succeeding VOPs. The syntax of a compressed bitstream of an audiovisual object fulfills the *MPEG-4 System and Description Language*. MPEG-4 permits either the use of machine-independent bytecode or the use of scripts. The bytecode approach can be used when the assumptions, on the templates to be described, are limited. Scripts are less flexible but are a concise approach to represent templates.

### MPEG-7 Standard

MPEG-7 [25] has the aim of defining a standard set of descriptors of multimedia information. In particular, MPEG-7 introduces the standardization of structures (*description scheme*) for the descriptors and their relationships. Descriptors and description schemes are associated with the multimedia content to permit effective searching. Description schemes can be hierarchical and multidimensional and can include images, video, graphics, audio, speech and textual annotations. MPEG-7 permits having different level of abstraction, from the lowest to the highest. For instance, if data are visual (e.g. images, videos), the lowest abstraction level can be a description of shape, texture, color, motion. The highest level covers semantic information. The highest level of description consists in the semantic information. Descriptions can vary on the basis of the data types and of the application context. Finally, MPEG-7 can address applications which can be stored on-line or off-line or streamed and can operate either in real-time or not critical time environments.

### MPEG-21 Standard

In the MPEG family, *MPEG-21* (also called *MPEG-21 Multimedia Framework*) is the newest proposal and became a standard at the end of 2003. It has the aim of enabling transparent and increased use of multimedia resources across a wide range of networks and devices. MPEG defines a *framework to support transactions that are interoperable and highly automated, specifically taking digital rights management (DRM) requirements and targeting multimedia access and delivery using heterogeneous network and terminals* [4]. More precisely, MPEG-21 aims to define a normative open framework for multimedia delivery and consumption for use by all the actors (e.g. content creators, providers, users) in the delivery and consumption chain. For this reason, MPEG-21 pays particular attention to *intellectual property management and protection (IPMP)* topics.

## 3.8 Conclusions

This chapter has presented image and video acquisition, representation and storage. Firstly, we have described human eye physiology paying attention to human color perception. Then we have described the structure of digital image acquisition devices. We have discussed the color representation in the digital images presenting the main color models used in image processing. Regarding storage, we have presented the main image formats describing JPEG in detail. Finally, we have reviewed video principles and the MPEG standard.

We conclude the chapter providing some bibliographical remarks. A comprehensive survey of the color representation can be found in [30]. JPEG standard is described in detail in [1]. The MPEG standards are fully discussed in [17][19][25].

## Problems

- 3.1.** Show that in the XYZ model the white is represented by the triple (1,1,1).
- 3.2.** Consider the YIQ model. Show that in a grayscale image, where  $R=G=B$ , the chrominance components  $I$  and  $Q$  are null.
- 3.3.** Consider the HSV model. Show that in the simplest form of HSV transformation, the hue ( $H$ ) become undefined when the saturation  $S$  is null.
- 3.4.** Compute in HSV model, the coordinates of cyan, magenta and yellow.
- 3.5.** Repeat Problem 3.4 for the HSB model.
- 3.6.** Take a videocassette registered under the NTSC system. How will it be displayed by a PAL videocassette recorder (VCR)? Explain your answer.
- 3.7.** Implement the Huffman coding algorithm. Test the software on the following example: consider a file formed by 10,000  $A$ , 2,000  $B$ , 25,000  $C$ , 5,000  $D$ , 40,000  $E$ , 18,000  $F$ . Compute how many bits are required to code the file.
- 3.8.** Consider the file formed by 20,000  $B$ , 2,500  $C$ , 50,000  $D$ , 4,000  $E$ , 1,800  $F$ . Compare, in terms of memory required, fix-length and Huffman coding. Does there exist a case where fix-length and Huffman coding require the same memory resources? Explain your answer.
- 3.9.** How much memory is required to store the movie *Casablanca* in its uncompressed version? Assume that the movie is black/white, has 25 frame/sec (each frame is  $640 \times 480$  pixels), its runtime is 102 minutes. For sake of simplicity, do not consider the memory required to store the audio of the movie.
- 3.10.** Repeat the Problem 3.9 for the movie *Titanic*. Titanic is a color movie, has 30 frame/sec, and its runtime is 194 minutes.

---

## References

1. T. Acharaya and A. K. Ray. *Image Processing: Principles and Applications*. John Wiley and Sons, 2005.
2. D. Ballard and C Brown. *Computer Vision*. Academic Press, 1982.
3. B. E. Bayer. Color imaging array. US Patent 3,971,065. Technical report, Eastman Kodak Company, 1976.
4. J. Bormans, J. Gelissen, and A. Perkis. MPEG-21: The 21<sup>st</sup> century multimedia framework. *IEEE Signal Processing Magazine*, 2003.
5. G. Buchsbaum. An analytical derivation of visual nonlinearity. *IEEE Transactions on Biomedical Engineering*, BME-27(5):237–242, 1980.
6. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
7. A. Del Bimbo, editor. *Visual Information Retrieval*. Morgan Kaufman Publishers, 1999.
8. T. Ebrahimi. MPEG-4 video verification model: A video encoding/decoding algorithm based on content representation. *Image Communication Journal*, 9(4):367–384, 1996.
9. K. S. Gibson and D. Nickerson. Analysis of the munsell colour system based on measurements made in 1919 and 1926. *Journal of Optical Society of America*, 3(12):591–608, 1940.
10. R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison Wesley, 1992.
11. G. Healey and Q. Luong. Color in computer vision: Recent progress. In *Handbook of Pattern Recognition and Computer Vision*, pages 283–312. World Scientific Publishing, 1998.
12. D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
13. L. M. Hurvich and D. Jameson. An opponent process theory of colour vision. *Psychological Review*, 64(6):384–404, 1957.
14. L. M. Hurvich and D. Jameson. Some quantitative aspects of an opponent-colors theory: Iv a psychological color specification system. *Journal of the Optical Society of America*, 45(6):416–421, 1957.
15. A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, 1989.
16. D. B. Judd and G. Wyszecki. *Color in Business, Science and Industry*. John Wiley and Sons, 1975.

17. R. Koenen, F. Pereira, and L. Chiariglione. MPEG-4: Context and objectives. *Image Communication Journal*, 9(4):295–304, 1997.
18. E. H. Land. Color vision and the natural images. *Proceedings of the National Academy of Sciences*, 45(1):116–129, 1959.
19. D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.
20. G. W. Meyer. Tutorial on colour science. *The Visual Computer*, 2(5):278–290, 1986.
21. A. H. Munsell. *An Atlas of the Munsell System*. Wassworth-Howland, 1915.
22. C. L. Novak and S. A. Shafer. *Color Vision. Encyclopedia of Artificial Intelligence*. John Wiley and Sons, 1992.
23. W. K. Pratt. *Digital Image Processing*. John Wiley and Sons, 1991.
24. T. Sakamoto, C. Nakanishi, and T. Hase. Software pixel interpolation for digital still cameras suitable for a 32-bit mcu. *IEEE Transactions on Consumer Electronics*, 44(4):1342–1352, 1998.
25. P. Salembier and J. R. Smith. MPEG-7 multimedia description schemes. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):748–759, 2001.
26. A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, 2001.
27. E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, 1998.
28. P. Tsai, T. Acharaya, and A. K. Ray. Adaptive fuzzy color interpolation. *Journal of Electronic Imaging*, 11(3):293–305, 2002.
29. B. A. Wandell. *Foundations of Vision*. Sinauer Associates, 1995.
30. G. Wyszecki and W. S. Stiles. *Color Science*. Mc Graw-Hill, 1982.

**Machine Learning**





## Machine Learning

### What the reader should know after reading this chapter

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.

### 4.1 Introduction

The ability to learn is one of the distinctive attributes of intelligent behavior. Following a seminal work [5], we can say that "*Learning process includes the acquisition of new declarative knowledge, the development of motor and cognitive skills through instruction or practice, the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation.*"

The study and computer modeling of learning processes in their multiple manifestations constitutes the topic of *machine learning*. Machine learning has been developed around the following primary research lines:

- *Task-oriented studies*, i.e. the development of learning systems to improve performance in a predetermined set of tasks.
- *Cognitive simulation*, namely, the investigation and computer simulation of human learning processes.
- *Theoretical analysis*, i.e. the theoretical investigation of possible learning methods and algorithms independently of applicative domain.

Machine learning methods, described in this book, are mainly the results of the first and third research lines.

The aim of this section is to provide a taxonomy of machine learning research, paying special attention to methods of learning by example.

The chapter is organized as follows: Section 4.2 provides a taxonomy of machine learning; in Section 4.3 learning by examples is discussed; finally, some conclusions are drawn in Section 4.4.

## 4.2 Taxonomy of Machine Learning

This section presents a taxonomy of machine learning, presenting useful criteria for classifying and comparing most machine learning investigations. Although machine learning systems can be classified according to different viewpoints [5], a common choice is to classify machine learning systems on the basis of the *underlying learning strategies* used.

In machine learning two entities, the *teacher* and the *learner*, play a crucial role. The teacher is the entity that has the required knowledge to perform a given task. The learner is the entity that has to learn the knowledge to perform the task.

We can distinguish learning strategies by the amount of inference the learner performs on the information provided by the teacher. We consider the two extreme cases, namely performing no inference and performing a remarkable amount of inference. If a computer system (the learner) is programmed directly, its knowledge increases but it performs no inference since all cognitive efforts are developed by the programmer (the teacher). On the other hand, if a system independently discovers new theories or invents new concepts, it must perform a very substantial amount of inference; it is deriving organized knowledge from experiments and observations. An intermediate case could be a student determining how to solve a math problem by analogy to problem solutions contained in a textbook. This process requires inference but much less than discovering a new theorem in mathematics.

Increasing the amount of inference that the learner is capable of performing, the burden on the teacher decreases. The taxonomy of machine learning below tries to capture the notion of trade-off in the amount of effort required of the learner and of the teacher [5]. Hence we can identify four different learning types: *rote learning*, *learning from instruction*, *learning by analogy* and *learning from examples*. The first three learning types are described below, while the next section is devoted to the last type.

### 4.2.1 Rote Learning

Rote learning consists in the direct implanting of new knowledge in the learner. No inference or other transformation of the knowledge is required on the part of the learner. Variants of this method include:

- Learning by being programmed or modified by an external identity. It requires no effort on the part of the learner. For instance, the usual style of computer programming.
- Learning by memorization of given facts and data with no inferences drawn from the incoming information. For instance, the primitive database systems.

### 4.2.2 Learning from Instruction

Learning from instruction (or *learning by being told*) consists in acquiring knowledge from a teacher or other organized source, such as a textbook, requiring that the learner transform the knowledge from the input language to an internal representation. The new information is integrated with prior knowledge for effective use. The learner is required to perform some inference, but a large fraction of the cognitive burden remains with the teacher, who must present and organize knowledge in a way that incrementally increases the learner's actual knowledge. Learning from instruction mimics education methods. Therefore, the machine learning task is to build a system that can accept instruction and can store and apply this learned knowledge effectively. Systems that use learning from instructions are described in [6][11][12].

### 4.2.3 Learning by Analogy

Learning by analogy consists in acquiring new facts or skills by transforming and increasing existing knowledge that bears strong similarity to the desired new concept or skill into a form effectively useful in the new situation. A learning-by-analogy system might be applied to convert an existing computer program into one that performs a closely related function for which it was not originally designed. Learning by analogy requires more inference on the part of the learner than rote learning or learning from instruction. A fact or skill analogous in relevant parameters must be retrieved from memory; then the retrieved knowledge must be transformed, applied to the new situation, and stored for future use. Systems that use learning by analogy are described in [1][4].

## 4.3 Learning from Examples

Given a set of examples of a concept, the learner induces a general concept description that describe the examples. The amount of inference performed by the learner is much greater than in learning from instruction and in learning by analogy. Learning from examples has become so popular in the last years that it is often called simply *learning*. In a similar way, the learner and examples are respectively referred as *learning machine* and *data*. In the rest of the book these conventions will be adopted.

The *learning problem* can be described as finding a general rule that explains data given only a sample of limited size. The difficulty of this task is similar to the problem of children learning to speak from the sounds emitted by the adults.

The learning problem can be stated as follows: given an example set of limited size, *find a concise data description*. Learning techniques can be grouped in three big families: *supervised learning*, *reinforcement learning* and *unsupervised learning*.

### 4.3.1 Supervised Learning

In supervised learning (or *learning with a teacher*), the data is a sample of input-output patterns. In this case, a concise description of the data is the *function* that can yield the output, given the input. This problem is called supervised learning because the objects under considerations are already associated with target values, e.g. classes and real values. Examples of this learning task are the recognition of handwritten letters and digits, the prediction of stock market indexes. Supervised algorithms are discussed in Chapters 8, 9 and 10.

In the problem of supervised learning, given a sample of input-output pairs, called the *training sample* (or *training set*), the task is to find a deterministic function that maps any input to an output that can predict future input-output observations, minimizing the errors as much as possible. Whenever asked for the target value of an object present in the training sample, it can return the value that appeared the highest number of times together with this object in the training sample. According to the type of the outputs, supervised learning can be distinguished in *classification* and *regression* learning.

#### Classification Learning

If the output space has no structure except whether two elements of the output are equal or not, this is called the problem of *classification learning* (or simply *classification*). Each element of the output space is called a *class*. The learning algorithm that solves the classification problem is called the *classifier*. In classification problems the task is to assign new inputs to one of a number of discrete classes or categories. This problem characterizes most pattern recognition tasks. A typical classification problem is to assign to a character bitmap the correct letter of the alphabet.

#### Regression

If the output space is formed by the values of continuous variables, for instance the stock exchange index at some future time, then the learning task is known as the problem of *regression* or *function learning* [7]. Typical examples of regression are to predict the value of shares in the stock exchange market and to estimate the value of a physical measure (e.g. pressure, temperature) in a section of a thermoelectric plant.

### 4.3.2 Reinforcement Learning

Reinforcement learning has its roots in control theory. It considers the scenario of a dynamic environment that results in state-action-reward triples as the data. The difference between reinforcement and supervised learning is that in reinforcement learning no optimal action exists in a given state, but

the learning algorithm must identify an action in order to maximize the expected reward over time. The concise description of data is the strategy that maximizes the reward.

The problem of reinforcement learning is to learn what to do, i.e. how to map situations to actions, in order to maximize a given reward. Unlike a supervised learning task, the learning algorithm is not told which actions to take in a given situation. Instead, the learner is assumed to gain information about the actions taken by some reward not necessarily arriving immediately after the action is taken. An example of such a problem is learning to play chess. Each board configuration, namely the position of chess pieces on the chess board, is a given state; the actions are the possible moves in a given configuration. The reward for a given action (e.g. the move of a piece), is winning the game. On the contrary, the punishment is losing the game. This reward, or this punishment, is delayed, which is very typical for reinforcement learning. Since a given state has no optimal action, one of the biggest challenges of a reinforcement learning algorithm is to find a trade-off between exploration and exploitation. In order to maximize reward (or minimize the punishment) a learning algorithm must choose actions which have been tried out in the past and found to be effective in producing reward, i.e. it must exploit its current knowledge. On the other hand, to discover those actions the learning algorithm has to choose actions not tried in the past and thus explore the state space. There is no general solution to this dilemma, but that neither of the two options can lead exclusively to an optimal strategy is clear.

A comprehensive survey on reinforcement learning can be found in [13].

### 4.3.3 Unsupervised Learning

If the data is only a sample of objects without associated target values, the problem is known as *unsupervised learning*. In unsupervised learning there is no teacher. Hence a concise description of the data can be a set of clusters or a probability density stating how likely it is to observe a certain object in the future. Typical examples of unsupervised learning tasks include the problem of image and text segmentation and the task of novelty detection in process control.

In unsupervised learning we are given a training sample of objects (e.g. images) with the aim of extracting some *structure* from them. For instance, identifying indoor or outdoor images or extracting face pixels in an image. If some structure exists in training data, it can take advantage of the redundancy and find a short description of data. A general way to represent data is to specify a similarity between any pairs of objects. If two objects share much structure, it should be possible to reproduce the data from the same prototype. This idea underlies *clustering algorithms* that form a rich subclass of unsupervised algorithms.

Clustering algorithms are based on the following idea. Given a fixed number of clusters, we aim to find a grouping of the objects such that similar

objects belong to the same cluster. If it is possible to find a clustering such that the similarities of the objects in one cluster are much greater than the similarities among objects from different clusters, we have extracted structure from the training sample so that the whole cluster can be represented by one representative data point. Clustering algorithms are discussed in detail in the Chapter 5.

In addition to clustering algorithms, in unsupervised learning techniques there are algorithms whose aim is to represent high-dimensionality data in low-dimension spaces, trying to preserve the original information of data. These techniques, called *dimensionality reduction methods (DRM)* are particular important for the following reasons. The use of more dimensions than strictly necessary leads to several problems. The first one is the space needed to store the data. As the amount of available information increases, the compression for storage purposes becomes even more important. The speed of algorithms using the data depends on the dimension of the vectors, so a reduction of the dimension can result in reduced computation time. Then it can be hard to make reliable classifiers when the dimensionality of input data is high (*curse of dimensionality* [2]). Curse of dimensionality and dimensionality reduction methods are described in Chapter 10.

## 4.4 Conclusions

In this chapter we have provided a taxonomy of machine learning research. We have discussed in detail learning by examples, topic of this book, introducing supervised and unsupervised learning. Finally, we conclude the chapter providing some bibliographical remarks, paying attention to the works who discuss machine learning in general. Machine learning has been discussed in detail for the first time in [8][9]. A modern approach to machine learning is discussed in [10]. Recent books (e.g. [3]), including this one, are focused essentially on learning by examples.

---

## References

1. J.R. Anderson. Acquisition of proof skills in geometry. In *Machine Learning*, pages 191–220. Tioga Publishing Company, 1983.
2. R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
3. C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
4. J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In *Machine Learning*, pages 137–162. Tioga Publishing Company, 1983.
5. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell. An overview of machine learning. In *Machine Learning*, pages 3–23. Tioga Publishing Company, 1983.
6. N. Haas and G.G. Hendrix. Learning by being told: Acquiring knowledge for information management. In *Machine Learning*, pages 405–428. Tioga Publishing Company, 1983.
7. R. Herbrich. *Learning Kernel Classifiers*. MIT Press, 2003.
8. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell. *Machine Learning*. Tioga Publishing Company, 1983.
9. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell. *Machine Learning*. Morgan Kaufman Publishers, 1986.
10. T. Mitchell. *Machine Learning*. Mc Graw-Hill, 1997.
11. D.J. Mostow. Machine transformation of advice into a heuristic search procedure. In *Machine Learning*, pages 367–404. Tioga Publishing Company, 1983.
12. M.D. Rychener. The instructible production system: A retrospective analysis. In *Machine Learning*, pages 429–459. Tioga Publishing Company, 1983.
13. R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.





## Bayesian Theory of Decision

### What the reader should know to understand this chapter

- Basic notions of statistics and probability theory (see Appendix A).
- Calculus notions are an advantage.

### What the reader should know after reading this chapter

- Basic notions of Bayesian theory (e.g., likelihood, priors, evidence).
- Fundamental concepts of the Bayesian theory of decision (e.g., loss function, Bayes decision rule).
- Discriminant functions.
- Normal distribution function.
- Whitening transformation.
- Receiver operating characteristic (ROC) curves.

## 5.1 Introduction

*Bayesian theory of decision (BTD)* is a fundamental tool of analysis in Machine Learning. Several machine learning algorithms have been derived using BTD. The fundamental idea in BTD is that the decision problem can be solved using probabilistic considerations. In order to introduce the theory we consider the following example. We suppose to have a classroom in which there are students of both genders. Moreover, there is an examiner, outside the classroom, that has to call the students for the examination. He has a list of the surnames of the students, but the surnames are not accompanied by the first names. How can the examiner decide if to a given surname corresponds a girl or a boy?

The aim of this chapter is to answer this question by introducing BTD. We will show that BTD is a *formalization of the common sense* [7]. There are many works on BTD [2][3][5][8][16][15], this chapter is inspired by the work

of [7], that represents a milestone in the history of pattern recognition and machine learning.

The chapter is organized as follows: Sections 5.2 and 5.3 present Bayes decision rule and Function respectively. Section 5.4 introduces the loss function; the special case of zero-one loss function is discussed in Section 5.5. Section 5.6 reviews discriminant functions; Section 5.7 introduces normal density and Whitening transform. In Section 5.8 we discuss the discriminant function when the likelihood assumes a normal distribution. Section 5.9 introduces Receiver Operating Curves. In Section 5.10 some conclusions are drawn; finally some problems are proposed at the end of the chapter.

## 5.2 Bayes Decision Rule

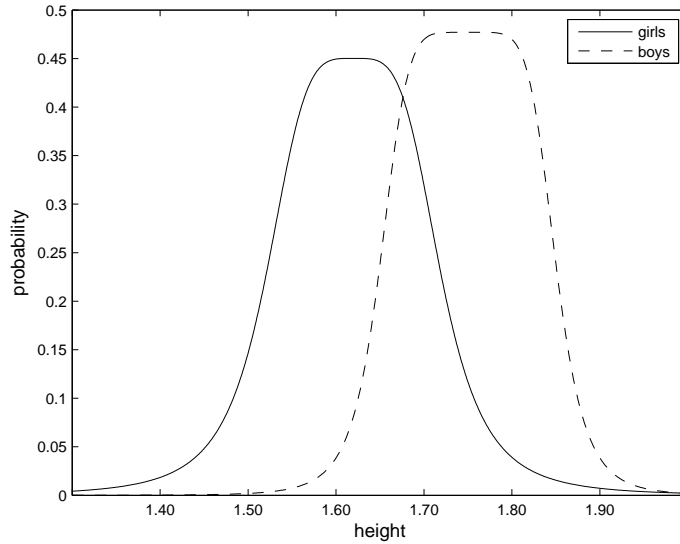
In this section we formalize what we have shown in the Introduction. We consider again our classroom with boys and girls and the examiner that has only a list with the surnames of the students. When the examiner calls a student (e.g. Smith) and the student appears, in decision-theoretic terminology we say that the student replies the *nature* in one of two possible states, i.e either the student is a boy or the student is a girl. We identify the *state of nature* (or *class*) with  $\mathcal{C}$ . If the student is a girl  $\mathcal{C} = \mathcal{C}_1$ , otherwise  $\mathcal{C} = \mathcal{C}_2$ . Since the state of nature is unknown a natural choice is to describe  $\mathcal{C}$  in a probabilistic way.

We assume that there is *prior probability*  $p(\mathcal{C}_1)$  that the student called by the examiner is a girl and  $p(\mathcal{C}_2)$  that is a boy. The sum of the prior probability over all possible classes, i.e.  $\mathcal{C}_1$  and  $\mathcal{C}_2$  in our example, must be one. If our examiner has to decide if the student Smith is a girl or a boy, in absence of further information he is forced to base his decision on prior probabilities. Hence he has to apply the following *decision rule*.

**Definition 1 (Prior Probability Decision Rule)** *Decide  $\mathcal{C}_1$  if  $p(\mathcal{C}_1) > p(\mathcal{C}_2)$ ; decide  $\mathcal{C}_2$  otherwise.*

If the amount of boys and girls is roughly the same, the previous decision rule will behave as the coin toss, i.e. it will be right only in half of the cases.

We suppose that the examiner for each student knows  $n$  numeric measurements (or *features*)  $\mathbf{x} = (x_1, \dots, x_n)$ , where, for instance,  $x_1$  is the height,  $x_2$  is the weight and so on. For sake of simplicity we suppose that the features are two and that are height and weight. For instance, if the height and the weight of the student Smith are, respectively,  $1.60m$  and  $59Kg$ , Smith can be represented by the *feature vector*  $(1.60, 59)$ . Generalizing we say that each student can be represented by a feature vector (or a *pattern*)  $\mathbf{x}$ . Since different features, as shown by the distribution of students' height in Figure 5.1, are associated to different students we can model the feature vector  $\mathbf{x}$  as a random variable whose distribution  $p(\mathbf{x}|\mathcal{C})$  depends on the state of nature  $\mathcal{C}$ . The distribution  $p(\mathbf{x}|\mathcal{C})$  is the *class-conditional probability density function*, i.e. the probability density function for  $\mathbf{x}$  when the state of the nature is  $\mathcal{C}$ .



**Fig. 5.1.** Hypothetical distribution of students' height in the classroom.

The set of pairs  $(\mathcal{X}, \mathcal{C}) = \{(\mathbf{x}_1, \mathcal{C}_1), (\mathbf{x}_2, \mathcal{C}_2), \dots, (\mathbf{x}_\ell, \mathcal{C}_\ell)\}$ , where the generic  $(\mathbf{x}_i, \mathcal{C}_i)$  means that  $\mathcal{C}_i$  is the state of nature of  $\mathbf{x}_i$ , is called simply *data* (or a *data set*). In the rest of the chapter we assume that data are *i.i.d* that stands for *independent and identically distributed* random variables. Saying that data are i.i.d. means that they are drawn independently according to the probability density  $p(\mathbf{x}|\mathcal{C})$ .

BTD assumes that all the relevant probability values are known, namely we assume that the prior probabilities  $p(\mathcal{C}_1)$ ,  $p(\mathcal{C}_2)$  and the class-conditional probability densities  $p(\mathbf{x}|\mathcal{C}_1)$ ,  $p(\mathbf{x}|\mathcal{C}_2)$  are known. The joint probability density of finding a pattern  $\mathbf{x}$  in the class  $p(\mathcal{C}_j)$  is:

$$p(\mathcal{C}_j, \mathbf{x}) = p(\mathcal{C}_j|\mathbf{x})p(\mathbf{x}). \quad (5.1)$$

But the same joint probability can also be written:

$$p(\mathcal{C}_j, \mathbf{x}) = p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j). \quad (5.2)$$

Plugging Equation (5.2) in (5.1) we get:

$$p(\mathcal{C}_j|\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j). \quad (5.3)$$

Dividing by  $p(\mathbf{x})$ , we finally get:

$$p(\mathcal{C}_j|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}{p(\mathbf{x})}. \quad (5.4)$$

We have proved the *Bayes Theorem* [1]. Equation (5.4) is called the *Bayes formula*.

The terms  $p(\mathcal{C}_j)$  and  $p(\mathcal{C}_j|\mathbf{x})$  are called respectively *prior probability* and *a posteriori probability*. The prior probability (or simply *prior*) expresses the a priori knowledge that we have on the problem, for instance the overall percentage of girls in the classroom. The a posteriori probability (or simply *posterior*) expresses the probability that the state of nature is  $\mathcal{C}_j$  when the pattern  $\mathbf{x}$  has been observed. The term  $p(\mathbf{x})$  is called *evidence* and in the case of two classes is:

$$p(\mathbf{x}) = \sum_{j=1}^2 p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j). \quad (5.5)$$

Evidence can be viewed as a normalization factor ensuring that the sum of the probabilities is one. Therefore evidence can be neglected and we can conclude that posterior probability is determined by the product  $p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)$ . When  $p(\mathbf{x}|\mathcal{C}_j)$  is large it is *likely* that the sample  $\mathbf{x}$  belongs to the class  $\mathcal{C}_j$ . Therefore the term  $p(\mathbf{x}|\mathcal{C}_j)$  is called the *likelihood of  $\mathcal{C}_j$  with respect to  $\mathbf{x}$* .

We consider a pattern  $\mathbf{x}$  for which  $p(\mathcal{C}_1|\mathbf{x})$  is larger than  $p(\mathcal{C}_2|\mathbf{x})$ , it is natural to decide that the pattern  $\mathbf{x}$  belongs to the class  $\mathcal{C}_1$ ; otherwise we assign the pattern to the class  $\mathcal{C}_2$ . It is quite easy to show that our strategy is theoretically correct. We observe that for a pattern  $\mathbf{x}$  the probability of error  $p(error|\mathbf{x})$  is  $p(\mathcal{C}_1|\mathbf{x})$  if we assign  $\mathbf{x}$  to  $\mathcal{C}_2$  (i.e.  $\mathbf{x} \in \mathcal{C}_2$ ), vice versa is  $p(\mathcal{C}_2|\mathbf{x})$  if we assign the pattern to  $\mathcal{C}_1$  (i.e.  $\mathbf{x} \in \mathcal{C}_1$ ). We can minimize  $p(error|\mathbf{x})$  by deciding  $\mathcal{C}_1$  if  $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x})$  and  $\mathcal{C}_2$  vice versa. Now we can compute the average probability of error  $p(error)$ :

$$p(error) = \int_{-\infty}^{\infty} p(error, \mathbf{x}) = \int_{-\infty}^{\infty} p(error|\mathbf{x})p(\mathbf{x})d(\mathbf{x}). \quad (5.6)$$

If we guarantee, deciding  $\mathcal{C}_1$  if  $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x})$  and  $\mathcal{C}_2$  otherwise, that  $p(error|\mathbf{x})$  is as small as possible, then  $p(error)$  has to be as small as possible. Hence the following *Bayes decision rule*

$$\text{Decide } \mathcal{C}_1 \text{ if } p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x}); \text{ otherwise decide } \mathcal{C}_2 \quad (5.7)$$

is justified.

The probability error  $P(error|\mathbf{x})$  associated to Bayes decision rule is:

$$P(error|\mathbf{x}) = \min(p(\mathcal{C}_1|\mathbf{x}), p(\mathcal{C}_2|\mathbf{x})). \quad (5.8)$$

Plugging equation (5.4) in (5.7) we get

$$\text{Decide } \mathcal{C}_1 \text{ if } \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} > \frac{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}{p(\mathbf{x})}; \text{ otherwise decide } \mathcal{C}_2.$$

Since the evidence  $p(\mathbf{x})$  is a normalization factor it can be neglected. Therefore we obtain

$$\text{Decide } \mathcal{C}_1 \text{ if } p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) > p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2); \text{ otherwise decide } \mathcal{C}_2. \quad (5.9)$$

### 5.3 Bayes Classifier\*

In this subsection we show formally that the Bayes decision rule is optimal. Following [6] we call an *observation* (or a feature vector or a pattern) a  $n$ -dimensional vector  $\mathbf{x}$  while its state of nature (or *class*)  $\mathcal{C}$ , takes value in a finite set  $[1, \dots, M]$ . This means that each pattern can have  $m$  different states of nature. The aim of machine learning is to build a mapping (or a *classifier*)  $\alpha : \mathbb{R}^n \rightarrow [1, M]$  which represents the guess of  $\mathcal{C}$  given  $\mathbf{x}$ . The classifier makes an error if  $\alpha(\mathbf{x}) \neq \mathcal{C}$ .

Let  $(X, \mathcal{Y})$  be a  $\mathbb{R}^n \times \{1, \dots, M\}$ -valued random pair. The distribution of  $(X, \mathcal{Y})$  describes the frequency of encountering particular pairs. An *error* occurs if  $\alpha(X) \neq \mathcal{Y}$  and the *probability of error* for  $\alpha$  is

$$L(\alpha) = P(\alpha(X) \neq \mathcal{Y}).$$

The best classifier  $\alpha^*$  is defined by

$$\alpha^* = \arg \min_{\alpha} P(\alpha(X) \neq \mathcal{Y}) \quad (5.10)$$

The mapping  $\alpha^*$  depends upon the distribution of  $(X, \mathcal{Y})$ . Therefore, if the distribution is known  $\alpha^*$  may be computed. The problem of finding  $\alpha^*$  is called the *Bayes problem*. The classifier  $\alpha^*$  is called the *Bayes classifier*. The minimal probability of error is called *Bayes error* and is denoted by  $L^* = L(\alpha^*)$ .

Now we pass to prove that Bayes classifier is optimal with respect to the error minimization. For the sake of simplicity, we suppose that the  $\mathcal{Y}$  assumes value in  $\{0, 1\}$  that corresponds to say that there are only two classes, the class '0' (i.e.  $\mathcal{Y} = 0$ ) and the class '1' (i.e.  $\mathcal{Y} = 1$ ). Given a  $n$ -dimensional vector  $\mathbf{x}$ , we define  $\eta(\mathbf{x})$  the conditional probability that  $\mathcal{Y}$  is 1 given  $X = \mathbf{x}$  such as:

$$\eta(\mathbf{x}) = P(\mathcal{Y} = 1 | X = \mathbf{x})$$

Any function  $\alpha : \mathbb{R}^n \rightarrow \{0, 1\}$  defines a *classifier* (or a *decision function*). Now, we define the *Bayes classifier*

$$\alpha^*(\mathbf{x}) = \begin{cases} 1 & \text{if } \eta(\mathbf{x}) > \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The following theorem shows that the Bayes classifier is optimal.

**Theorem 1 (Bayes Classifier Optimality)** *For any classifier  $\alpha : \mathbb{R}^n \rightarrow \{0, 1\}$ ,*

$$P(\alpha^*(X) \neq \mathcal{Y}) \leq P(\alpha(X) \neq \mathcal{Y}).$$

*that is, the Bayes classifier  $\alpha^*$  is the optimal classifier.*

*Proof*

Let  $X = \mathbf{x}$ , the conditional error probability  $P(\alpha(X) \neq \mathcal{Y} | X = \mathbf{x})$  of any  $\alpha$  is expressed by:

$$\begin{aligned}
&= 1 - P(\alpha(X) \neq \mathcal{Y} | X = \mathbf{x}) \\
&= 1 - P(\mathcal{Y} = 1, \alpha(X) = 1 | X = \mathbf{x}) - P(\mathcal{Y} = 0, \alpha(X) = 0 | X = \mathbf{x}) \\
&= 1 - [\mathbf{I}_{\alpha(\mathbf{x})=1} P(\mathcal{Y} = 1 | X = \mathbf{x}) + \mathbf{I}_{\alpha(\mathbf{x})=0} P(\mathcal{Y} = 0 | X = \mathbf{x})] \\
&= 1 - [\mathbf{I}_{\alpha(\mathbf{x})=1} \eta(\mathbf{x}) + \mathbf{I}_{\alpha(\mathbf{x})=0} (1 - \eta(\mathbf{x}))]
\end{aligned} \tag{5.11}$$

where  $\mathbf{I}^1$  is the indicator function.

Thus  $P(\alpha(X) \neq \mathcal{Y} | X = \mathbf{x}) - P(\alpha^*(X) \neq \mathcal{Y} | X = \mathbf{x})$  is given by:

$$\begin{aligned}
&= \eta(\mathbf{x})[\mathbf{I}_{\alpha^*(\mathbf{x})=1} - \mathbf{I}_{\alpha(\mathbf{x})=1}] + (1 - \eta(\mathbf{x}))[\mathbf{I}_{\alpha^*(\mathbf{x})=0} - \mathbf{I}_{\alpha(\mathbf{x})=0}] \\
&= \eta(\mathbf{x})[\mathbf{I}_{\alpha^*(\mathbf{x})=1} - \mathbf{I}_{\alpha(\mathbf{x})=1}] + (1 - \eta(\mathbf{x}))[\mathbf{I}_{\alpha(\mathbf{x})=1} - \mathbf{I}_{\alpha^*(\mathbf{x})=1}] \\
&= (2\eta(\mathbf{x}) - 1)[\mathbf{I}_{\alpha^*(\mathbf{x})=1} - \mathbf{I}_{\alpha(\mathbf{x})=1}] \\
&\geq 0
\end{aligned} \tag{5.12}$$

If  $\eta(\mathbf{x}) > \frac{1}{2}$  the first (by definition) and the second term<sup>2</sup> of (5.12) are nonnegative and their product is still nonnegative. On the other hand, if  $\eta(\mathbf{x}) \leq \frac{1}{2}$  the first and the second term<sup>3</sup> are nonpositive and their product is again nonnegative. Hence the theorem statement is proved.

## 5.4 Loss Function

In Section 5.2 we gave the expression of the probability error, in the case of two classes, associated with the Bayes decision rule. Now we generalize the approach considering more than two classes and defining *the loss function*. Intuitively, we can view the loss function as a tool to measure the performance of a decision algorithm (or *classifier*). This approach permits taking actions that are different from the usual classification, for instance the *rejection*. In some applications it is mandatory to minimize the error as much as possible. For instance, the maximum error that is acceptable for a postal OCR, that is, the device that reads automatically the address of a letter, cannot exceed 1.5%. Therefore, deciding the rejection (e.g. the classifier refuses to make a decision) when the the probability error is not acceptable is a correct policy.

The loss function measures the cost of each classifier action and converts an error probability error into a decision. This approach allows us to handle situations in which particular classification mistakes has to be considered differently from the others. For instance, classifying in a patient a malignant tumour as benign is heavier than classifying a benign tumor as malignant, since in the first case the patient is not to undergo therapy against the cancer.

<sup>1</sup>  $\mathbf{I}_{\alpha(\mathbf{x})=1}$  is 1 if  $\alpha(\mathbf{x}) = 1$ ; 0 otherwise.

<sup>2</sup> Since  $\mathbf{I}_{\alpha^*(\mathbf{x})=1}$  is 1, the term must be nonnegative

<sup>3</sup> Since  $\mathbf{I}_{\alpha^*(\mathbf{x})=1}$  is 0, the term must be nonpositive

Besides, we can decide that the cost of a misclassification of a pattern can depend by the *a priori* probability of the membership class. Namely the cost of misclassifying a pattern that belongs to a class  $i$  can be considered heavier if  $P(\mathcal{C}_i)$  is high. In some modern languages a few characters are very unusual (e.g. in Italian the  $q$  and in Greek the  $\xi$ ) hence the cost of misclassification of these character can be less heavy than the one associated to other characters, since the overall performance of an OCR is marginally affected by the misclassification of these characters. Now we pass to the formal description of the loss function.

Let  $(\mathcal{C}_1, \dots, \mathcal{C}_M)$  be the finite set of the possible classes the patterns belong to and let  $\mathcal{B} = (\beta_1, \dots, \beta_n)$  be the set of the possible action of the classifier. The loss function  $\pi(\beta_i|\mathcal{C}_j)$  measures the penalty (or *loss*) that the classifier receives when takes the action  $\beta_i$  and the pattern  $\mathbf{x}$  belongs to the  $\mathcal{C}_j$ . Let  $p(\mathbf{x}|\mathcal{C}_j)$  be the state-conditional probability density function for  $\mathbf{x}$  given that  $\mathcal{C}_j$  is the class the pattern belongs to. Hence remembering Bayes formula the posterior probability  $p(\mathcal{C}_j|\mathbf{x})$  is given by:

$$p(\mathcal{C}_j|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}{p(\mathbf{x})} \quad (5.13)$$

where the evidence is:

$$p(\mathbf{x}) = \sum_{j=1}^M p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j). \quad (5.14)$$

Now we consider a particular sample  $\mathbf{x}$  and we assume to take an action  $\beta_i$ . If the class the pattern belongs to is  $\mathcal{C}_j$ , the loss associated with the action is  $\pi(\beta_i|\mathcal{C}_j)$ . Hence the *expected loss*  $\mathcal{R}(\beta_i|\mathbf{x})$  associated with the action  $\beta_i$  is:

$$\mathcal{R}(\beta_i|\mathbf{x}) = \sum_{j=1}^M \pi(\beta_i|\mathcal{C}_j)p(\mathcal{C}_j|\mathbf{x}). \quad (5.15)$$

In machine learning an expected loss is called *risk* and the term  $\mathcal{R}(\beta_i|\mathbf{x})$  is called *conditional risk*.

When we observe a pattern  $\mathbf{x}$  we can minimize the risk by choosing the action that minimizes the conditional risk. Hence the problem of choosing the action can be viewed as to find a decision rule that minimizes the overall risk. Formally a *decision rule* is a function  $\beta(\mathbf{x})$  whose output is the action to take for every pattern  $\mathbf{x}$ . For every  $\mathbf{x}$  the output of  $\beta(\mathbf{x})$  is an element of the set  $\mathcal{B}$ .

Given a decision rule  $\beta(\mathbf{x})$ , the overall risk  $\mathcal{R}$  is:

$$\mathcal{R} = \int \mathcal{R}(\beta(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x}. \quad (5.16)$$

If we select  $\beta(\mathbf{x})$  so that the conditional risk is as small as possible for every  $\mathbf{x}$ , the overall risk is minimized. This justifies the following alternative definition of the Bayes decision rule:

**Definition 2 (Bayes Decision Rule)** *To minimize the overall risk, compute the conditional risk*

$$\mathcal{R}(\beta_i|\mathbf{x}) = \sum_{j=1}^M \pi(\beta_i|\mathcal{C}_j)p(\mathcal{C}_j|\mathbf{x}). \quad (5.17)$$

for  $i=1, \dots, n$  and then choose the action  $\beta_i$  for which  $\mathcal{R}(\beta_i|\mathbf{x})$  is minimum.

The minimum  $\mathcal{R}^*$  resulting, with the application of Bayes decision rule, is called *Bayes risk*.

#### 5.4.1 Binary Classification

In this subsection we apply the previous considerations to the special case of *binary classification*, e.g. a classification problem with only two classes. A classifier that assigns a pattern to one of two classes is called a *binary classifier* (or a *dichotomizer*). Whereas a classifier with more than two classes is called a *polychotomizer*.

In the case of binary classification, action  $\beta_1$  stands for deciding that the pattern  $\mathbf{x}$  belongs to the class  $\mathcal{C}_1$ , whereas action  $\beta_2$  stands for deciding that the pattern  $\mathbf{x}$  belongs to the class  $\mathcal{C}_2$ . The conditional risk, given by (5.17), in the binary classification is:

$$\mathcal{R}(\beta_1|\mathbf{x}) = \pi(\beta_1|\mathcal{C}_1)p(\mathcal{C}_1|\mathbf{x}) + \pi(\beta_1|\mathcal{C}_2)p(\mathcal{C}_2|\mathbf{x}) \quad (5.18)$$

$$\mathcal{R}(\beta_2|\mathbf{x}) = \pi(\beta_2|\mathcal{C}_1)p(\mathcal{C}_1|\mathbf{x}) + \pi(\beta_2|\mathcal{C}_2)p(\mathcal{C}_2|\mathbf{x}). \quad (5.19)$$

Hence Bayes decision rule in this case is

**Definition 3 (Bayes Decision Rule; Binary Classification)** *Decide  $\mathcal{C}_1$  if  $\mathcal{R}(\beta_1|\mathbf{x}) < \mathcal{R}(\beta_2|\mathbf{x})$ ; Decide  $\mathcal{C}_2$  otherwise.*

The same rule can be reformulated in terms of posterior probabilities

**Definition 4** *Decide  $\mathcal{C}_1$  if*

$$(\pi(\beta_2|\mathcal{C}_1) - \pi(\beta_1|\mathcal{C}_1))p(\mathcal{C}_1|\mathbf{x}) > (\pi(\beta_1|\mathcal{C}_2) - \pi(\beta_2|\mathcal{C}_2))p(\mathcal{C}_2|\mathbf{x}); \quad (5.20)$$

*Decide  $\mathcal{C}_2$  otherwise.*

The factors  $(\pi(\beta_2|\mathcal{C}_1) - \pi(\beta_1|\mathcal{C}_1))$  and  $(\pi(\beta_1|\mathcal{C}_2) - \pi(\beta_2|\mathcal{C}_2))$  are positive since the loss associated to an error is larger than the loss associated to a correct classification. Therefore the decision of the classifier is determined by what probability between  $p(\mathcal{C}_1|\mathbf{x})$  and  $p(\mathcal{C}_2|\mathbf{x})$  is larger.

If we apply the Bayes theorem at (5.20) we get

$$(\pi(\beta_2|\mathcal{C}_1) - \pi(\beta_1|\mathcal{C}_1))p(\mathbf{x}|\mathcal{C}_1)p(\mathbf{x}) > (\pi(\beta_1|\mathcal{C}_2) - \pi(\beta_2|\mathcal{C}_2))p(\mathbf{x}|\mathcal{C}_2)p(\mathbf{x}). \quad (5.21)$$

Assuming that  $(\pi(\beta_2|\mathcal{C}_1) - \pi(\beta_1|\mathcal{C}_1))$  is positive, that is correct since the loss associated to an error is larger the one associated to a correct classification, we can rearranging the terms of (5.21) obtaining the following expression:



$$\frac{p(\mathbf{x}|\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)} > \frac{(\pi(\beta_1|\mathcal{C}_2) - \pi(\beta_2|\mathcal{C}_2)) p(\mathcal{C}_1)}{(\pi(\beta_2|\mathcal{C}_1) - \pi(\beta_1|\mathcal{C}_1)) p(\mathcal{C}_2)}. \quad (5.22)$$

Hence an alternative expression of Bayes rule is:

**Definition 5** *Decide  $\mathcal{C}_1$  if the inequality (5.22) holds; Decide  $\mathcal{C}_2$  otherwise.*

The term  $\frac{p(\mathbf{x}|\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)}$  is called the *likelihood ratio*. Hence if the likelihood ratio exceeds a threshold, that does not depend by the pattern, the decision is  $\mathcal{C}_1$ , otherwise  $\mathcal{C}_2$ .

## 5.5 Zero-One Loss Function

In classification each pattern  $\mathbf{x}$  is associated to a class, and the action  $\beta_i$  of the classifier generally consists in deciding that the pattern belongs to a class  $\mathcal{C}_i$ . If the action  $\beta_i$  is taken and the pattern belongs, in nature, to the pattern  $\mathcal{C}_j$ ; the decision is correct if  $i = j$ , otherwise is an error. In order to find a decision rule that minimizes the error rate, the first step consists in looking for the loss function that is appropriate for the situation described above. The loss function is the so-called *symmetrical* or *zero-one loss function*

$$\pi(\beta_i|\mathcal{C}_j) = \left\{ \begin{array}{ll} 0 & i = j \\ 1 & i \neq j. \end{array} \right. \quad i, j = 1, \dots, M$$

This function assigns no penalty to a correct decision, vice versa any error has penalty one. In this way, all errors are evaluated in the same manner.

If we apply the zero-one loss function to the conditional risk, that is given by Equation (5.15), we get:

$$\begin{aligned} \mathcal{R}(\beta_i|\mathbf{x}) &= \sum_{j=1}^M \pi(\beta_i|\mathcal{C}_j) p(\mathcal{C}_j|\mathbf{x}) \\ &= \sum_{j \neq i}^M p(\mathcal{C}_j|\mathbf{x}) \\ &= 1 - p(\mathcal{C}_i|\mathbf{x}) \end{aligned} \quad (5.23)$$

where  $1 - p(\mathcal{C}_i|\mathbf{x})$  is the conditional probability that the action  $\beta_i$ , namely to assign the pattern  $\mathbf{x}$ , is correct.

The Bayes decision rule consists in choosing the action that minimizes the conditional risk. Since the conditional risk is given by Equation (5.23), minimizing the conditional risk corresponds to maximizing the a posteriori probability  $p(\mathcal{C}_i|\mathbf{x})$ . Hence the first formulation of Bayes decision rule, given in (5.9), is justified.

## 5.6 Discriminant Functions

The use of *discriminant functions* is a popular approach to make a classifier.

**Definition 6 (Discriminant Functions)** *Given a pattern  $\mathbf{x} \in \mathbb{R}^n$ , and the finite set of the possible classes  $\mathcal{C}^* = (\mathcal{C}_1, \dots, \mathcal{C}_M)$ , we call  $\mathcal{G} = (\gamma_1(\mathbf{x}), \dots, \gamma_M(\mathbf{x}))$  with  $\gamma_i : \mathbb{R}^n \rightarrow \mathbb{R}$ , a set of discriminant functions. The single function  $\gamma_i$  ( $i = 1, \dots, M$ ) is called a **discriminant function**.*

Using the set  $\mathcal{G}$  we can get the following *discriminant function rule*

**Definition 7 (Discriminant Function Rule)** *Assign the pattern  $\mathbf{x}$  to the class  $\mathcal{C}_i$  if*

$$\gamma_i(\mathbf{x}) > \gamma_j(\mathbf{x}) \quad \forall j \neq i. \quad (5.24)$$

If we want to make classifier, it is adequate to make a machine (e.g. a computer program or an hardware device) that computes the set of discriminant functions  $\mathcal{G}$  and chooses the class that corresponds to the function that assumes the highest value for the pattern  $\mathbf{x}$ .

Now we show that it is easy to represent a Bayes classifier in the framework of the discriminant functions. If for each conditional risk  $\mathcal{R}(\beta_i|\mathbf{x})$  we define a discriminant function  $\gamma_i(\mathbf{x}) = -\mathcal{R}(\beta_i|\mathbf{x})$ , choosing the maximum discriminant function implies the minimization of the corresponding conditional risk. Hence the discriminant function rule is an alternative way to the Bayes decision rule.

The set of discriminant functions is not uniquely determined. For instance, if we add each function with the same real constant we get a new set of discriminant functions which produces the same classifications produced by the former set. The same effect we obtain if we multiply each discriminant function by a positive constant. If we replace each discriminant function  $\gamma_i$  with a function  $\phi(\gamma_i)$  where  $\phi(\cdot)$  is a continuous monotonic increasing function, we obtain the same classifier.

Now we pass to compute the form of the set of discriminant functions when we use the zero-one loss function. In this case each discriminant function is given by:

$$\begin{aligned} \gamma_i(\mathbf{x}) &= -\mathcal{R}(\beta_i|\mathbf{x}) \\ &= p(\mathcal{C}_i|\mathbf{x}) - 1 \\ &= p(\mathcal{C}_i|\mathbf{x}). \end{aligned} \quad (5.25)$$

The last equality holds since we can ignore the subtraction of real constants. Applying the Bayes theorem (5.25) we get

$$\begin{aligned} \gamma_i(\mathbf{x}) &= p(\mathcal{C}_i|\mathbf{x}) \\ &= \frac{p(\mathbf{x}|\mathcal{C}_i)p(\mathcal{C}_i)}{p(\mathbf{x})}. \end{aligned} \quad (5.26)$$

If we take the logarithm in both sides of Equation (5.26) and we define  $\gamma'_i(\mathbf{x}) = \ln \gamma_i(\mathbf{x})$ , we get

$$\gamma'_i(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_i) + \ln p(\mathcal{C}_i) - \ln p(\mathbf{x}) \quad (5.27)$$

Since the evidence  $p(\mathbf{x})$  is a scalar, the term  $\ln p(\mathbf{x})$  can be neglected. Hence we obtain the final formula:

$$\gamma'_i(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_i) + \ln p(\mathcal{C}_i). \quad (5.28)$$

The use of a set of discriminant functions induces a partition of  $\mathbb{R}^n$ , which is divided into  $M$  *decision regions*,  $\mathcal{D}_1, \dots, \mathcal{D}_M$ .

If  $\gamma_i(\mathbf{x}) > \gamma_j(\mathbf{x}) \quad \forall j \neq i$  then  $\mathbf{x} \in \mathcal{D}_i$ . The decision regions are separated by *decision boundaries* that are hypersurfaces in  $\mathbb{R}^n$ .

### 5.6.1 Binary Classification Case

In this subsection we derive the set of discriminant function in the case of a binary classification, namely when the classes are two. When the classes are two we should use two discriminant functions  $\gamma_1(\mathbf{x})$  and  $\gamma_2(\mathbf{x})$  and assigning the pattern  $\mathbf{x}$  to  $\mathcal{C}_1$  if  $\gamma_1(\mathbf{x}) > \gamma_2(\mathbf{x})$ . An alternative approach consists in defining a unique discriminant function  $\gamma(\mathbf{x})$  that is the difference between two discriminant functions, namely

$$\gamma(\mathbf{x}) = \gamma_1(\mathbf{x}) - \gamma_2(\mathbf{x}). \quad (5.29)$$

If we use  $\gamma(\mathbf{x})$  the decision rule becomes:

*Decide  $\mathcal{C}_1$  if  $\gamma(\mathbf{x}) > 0$ ; Decide  $\mathcal{C}_2$  otherwise.*

If we plug Equation (5.25) in (5.29) we get the following expression:

$$\gamma(\mathbf{x}) = p(\mathcal{C}_1|\mathbf{x}) - p(\mathcal{C}_2|\mathbf{x}) \quad (5.30)$$

It is possible to obtain an alternative expression for the discriminant function if we apply (5.28) to (5.29):

$$\gamma(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_1) + \ln p(\mathcal{C}_1) - \ln p(\mathbf{x}|\mathcal{C}_2) - \ln p(\mathcal{C}_2) \quad (5.31)$$

$$= \ln \frac{p(\mathbf{x}|\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)} + \ln \frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}. \quad (5.32)$$

## 5.7 Gaussian Density

This section provides a brief description of the Gaussian probability density.

First of all, we recall a *probability density function* as a nonnegative function  $p: \mathbb{R} \rightarrow [0, 1]$  that fulfills the condition:

$$\int_{-\infty}^{\infty} p(x) dx = 1. \quad (5.33)$$

Then we define the *expected value* of a scalar function  $f(x)$  for some probability density function  $p(x)$ :

$$\mathcal{E}(f(x)) = \int_{-\infty}^{\infty} f(x)p(x)dx. \quad (5.34)$$

If  $x$  assumes values only on a discrete set  $\mathcal{S}$ , the expected value is

$$\mathcal{E}(f(x)) = \sum_{x \in \mathcal{S}} f(x)p(x). \quad (5.35)$$

### 5.7.1 Univariate Gaussian Density

The continuous *univariate Gaussian density* (or *univariate normal density*)  $p(x)$  is a probability density function defined by (see Figure 5.2):

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp \left[ -\frac{1}{2} \left( \frac{x - \mu}{\sigma} \right)^2 \right] \quad (5.36)$$

where  $\mu$  is the *expected value* (or *mean*) of  $x$  defined by

$$\mu = \mathcal{E}(x) = \int_{-\infty}^{\infty} xp(x)dx \quad (5.37)$$

and where  $\sigma^2$  is the *variable*

$$\sigma^2 = \mathcal{E}(x - \mu)^2 = \int_{-\infty}^{\infty} (x - \mu)^2 p(x)dx. \quad (5.38)$$

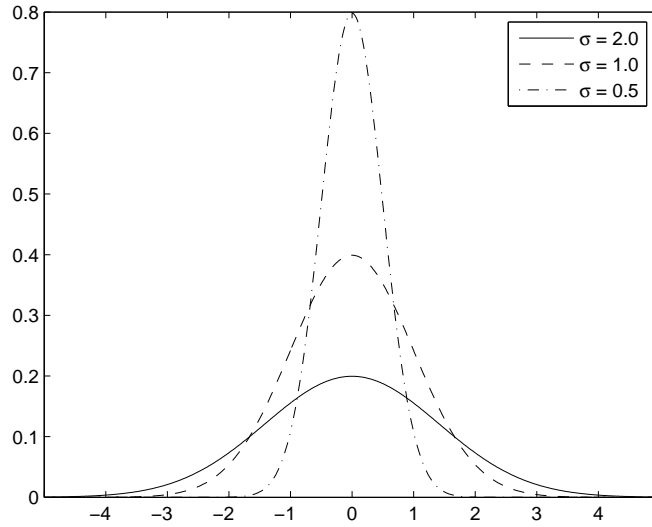
The Gaussian density is fully characterized by the mean  $\mu$  and the variance  $\sigma^2$ , therefore the Gaussian is often indicated with  $\mathcal{N}(\mu, \sigma^2)$ .

The importance of the Gaussian density is underlined by the following fact. The aggregate effect of the sum of a large number of independent random variables, leads to a normal distribution. Since patterns can be considered as ideal prototypes corrupted by a large number of random processes (e.g. noise), the Gaussian is usually a very good model to represent the probability distribution of the patterns.

### 5.7.2 Multivariate Gaussian Density

In this subsection the variable  $\mathbf{x}$  is *multivariate*, namely  $\mathbf{x}$  is a vector with  $n$  components ( $\mathbf{x} \in \mathbb{R}^n$ ). In this case the Gaussian density called *multivariate Gaussian density* (or *normal Gaussian density*)  $p(\mathbf{x})$  is given by (see example in Figure 5.3):

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] \quad (5.39)$$



**Fig. 5.2.** Gaussian curve for different value of  $\sigma$ .

where  $\mathbf{x}, \boldsymbol{\mu} \in \mathbb{R}^n$ ,  $\boldsymbol{\Sigma}$  is a  $n \times n$  covariance matrix.  $|\boldsymbol{\Sigma}|$  and  $\boldsymbol{\Sigma}^{-1}$  are, respectively, the determinant of the covariance matrix and its inverse;  $(\mathbf{x} - \boldsymbol{\mu})^T$  denote the transpose of  $(\mathbf{x} - \boldsymbol{\mu})$ .

The mean  $\boldsymbol{\mu}$  and the covariance matrix  $\boldsymbol{\Sigma}$  are given by:

$$\boldsymbol{\mu} = \mathcal{E}(x) = \int \mathbf{x}p(\mathbf{x})d\mathbf{x} \quad (5.40)$$

$$\boldsymbol{\Sigma} = \mathcal{E}(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T = \int (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T p(\mathbf{x})d\mathbf{x}. \quad (5.41)$$

The covariance matrix  $\boldsymbol{\Sigma}$  is always *symmetric* (i.e.  $\boldsymbol{\Sigma}_{ij} = \boldsymbol{\Sigma}_{ji} \forall i, j$ ) and *positive semidefinite*, that is all its eigenvalues  $\lambda_1, \dots, \lambda_n$  are nonnegative ( $\lambda_i \geq 0 \ i = 1, \dots, n$ ).

If  $x_i$  is the  $i$ -th component of  $\mathbf{x}$ ,  $\mu_i$  is  $i$ -th component of  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}_{ij}$  the  $ij$ -th component of  $\boldsymbol{\Sigma}$ , then

$$\mu_i = \mathcal{E}(x_i) \quad (5.42)$$

$$\boldsymbol{\Sigma}_{ij} = \mathcal{E}((x_i - \mu_i)(x_j - \mu_j)). \quad (5.43)$$

The diagonal elements of the covariance matrix  $\boldsymbol{\Sigma}_{ii}$  are the variances of  $x_i$ , i.e.  $\boldsymbol{\Sigma}_{ii} = \mathcal{E}((x_i - \mu_i)(x_i - \mu_i))$ . The other elements  $\boldsymbol{\Sigma}_{ij}$  (with  $i \neq j$ ) are the *covariances* of  $x_i$  and  $x_j$ . If  $x_i$  and  $x_j$  are *statistically independent*, then  $\boldsymbol{\Sigma}_{ij} = 0$ .

The quantity  $d^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$  is called the *squared Mahalanobis distance* between  $\mathbf{x}$  and  $\boldsymbol{\mu}$ .

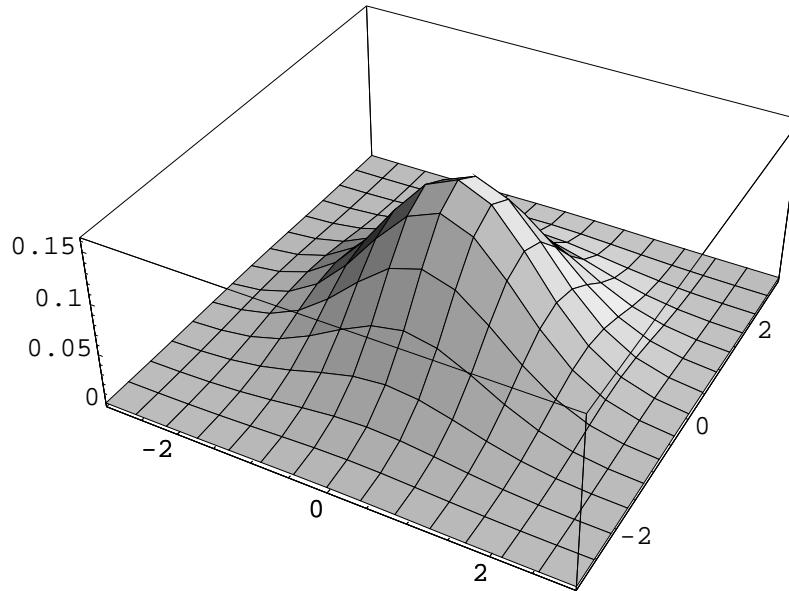


Fig. 5.3. Gaussian in two dimensions.

### 5.7.3 Whitening Transformation

In this subsection we introduce the whitening transformation, a very popular technique to preprocess the data. For instance, the whitening transformation is a basic tool in the *independent component analysis* [4][12] computation (see Chapter 11). Now we pass to introduce the whitening transformation.

Let  $\Omega = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  be a data set, formed by vectors  $\mathbf{x}_i \in \mathbb{R}^n$ , which has mean  $\langle \mathbf{x} \rangle$  and covariance matrix  $\Sigma$ . Then we introduce the eigenvalue equation

$$\Sigma U = U \Lambda \quad (5.44)$$

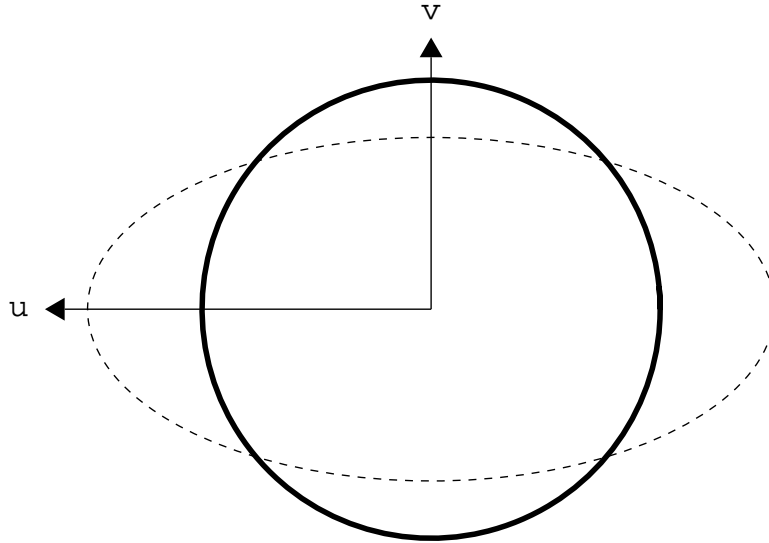
where  $U$  is a  $n \times n$  matrix, consisting of  $N$  eigenvectors as  $U = [u_1, \dots, u_n]$  and  $\Lambda$  is a diagonal matrix of eigenvalues as:

$$\begin{bmatrix} \lambda_1 & 0 & \cdots \\ 0 & \ddots & 0 \\ 0 & \cdots & \lambda_n \end{bmatrix}.$$

Then we can define a new transformation of data that maps the data matrix  $X$  into a new matrix  $Y$ , whose covariance matrix is the identity matrix  $\mathbb{I}$

$$Y = \Lambda^{-\frac{1}{2}} U^T X = (U \Lambda^{-\frac{1}{2}})^T X. \quad (5.45)$$

The transformation  $U \Lambda^{-\frac{1}{2}}$  is called the *whitening transformation* or the *whitening process*. The transformation  $U$  is the *Principal Component Analysis*



**Fig. 5.4.** Schematic illustration of the whitening transform in a bidimensional space. After PCA the data are distributed in an ellipse with semiaxes  $u$  and  $v$ , which are the eigenvectors of the covariance matrix. After the whitening transform the data are in a circle of unitary radius.

(PCA) [13], that projects the data along the directions of maximal variance i.e the *principal components* (see Chapter 11). The aim of the whitening transformation is to change the scales of the principal components in proportion to  $\frac{1}{\sqrt{\lambda_i}}$ . The effect of the whitening transformation is shown in Figure 5.4.

The following theorem [10] underlines basic properties of the whitening transformation:

**Theorem 2** *The whitening transformation*

- (i) *is not orthonormal*
- (ii) *does not preserve Euclidean distances*

*Proof*

- (i) The whitening transformation is not orthonormal since we have:

$$(U\Lambda^{-\frac{1}{2}})^T(U\Lambda^{-\frac{1}{2}}) = \Lambda^{-\frac{1}{2}}U^T U\Lambda^{-\frac{1}{2}} = \Lambda^{-1} \neq \mathbb{I}. \quad (5.46)$$

- (ii) Euclidean distances are not preserved, since we have:

$$\|Y\|^2 = Y^T Y = (\Lambda^{-\frac{1}{2}}U^T X)^T (\Lambda^{-\frac{1}{2}}U^T X) = X^T U \Lambda^{-1} U^T X \neq \|X\|^2. \quad (5.47)$$

## 5.8 Discriminant Functions for Gaussian Likelihood

In this section we investigate the discriminant functions in the special case that the likelihood  $p(\mathbf{x}|\mathcal{C}_i)$  assumes a Gaussian distribution. In Section 5.6 we have seen that the discriminant functions  $\gamma_i(\mathbf{x})$  can be represented by the following equation:

$$\gamma_i(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_i) + \ln p(\mathcal{C}_i). \quad (5.48)$$

If we suppose that the likelihood  $p(\mathbf{x}|\mathcal{C}_i)$  has a normal distribution, i.e.  $p(\mathbf{x}|\mathcal{C}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$  and we plug in (5.48), we get:

$$\gamma_i(\mathbf{x}) = -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \ln p(\mathcal{C}_i). \quad (5.49)$$

Now we discuss the form that (5.49) assumes in particular cases.

### 5.8.1 Features Are Statistically Independent

When the features are statistically independent, the non-diagonal elements of the covariance matrix  $\boldsymbol{\Sigma}$  are null. For sake of simplicity, we assume in addition that each feature  $x_i$  has the same variance  $\sigma^2$ . This assumption corresponds to the situation in which all patterns fall in hyperspherical clusters of equal size. Under this further condition, the covariance matrix  $\boldsymbol{\Sigma}$  is a multiple of the covariance matrix that is  $\boldsymbol{\Sigma} = \sigma^2 \mathbb{I}$ . Therefore the inverse and the determinant of the covariance matrix  $\neq$  are, respectively:

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{\sigma^2} \mathbb{I} \quad (5.50)$$

$$|\boldsymbol{\Sigma}| = \sigma^{2n}. \quad (5.51)$$

Substituting them in (5.49) we get:

$$\gamma_i(\mathbf{x}) = -\frac{d}{2} \ln 2\pi - n \ln \sigma - \frac{1}{2\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}\|^2 + \ln p(\mathcal{C}_i). \quad (5.52)$$

Since the first two terms are additive constants, we can neglect them obtaining:

$$\gamma_i(\mathbf{x}) = -\frac{1}{2\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 + \ln p(\mathcal{C}_i). \quad (5.53)$$

### Prior Probabilities Are All Equal

If the prior probability  $\mathcal{C}_i$  is the same for each class, it becomes an additive constant that can be neglected and becomes:

$$\gamma_i(\mathbf{x}) = -\frac{1}{2\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \quad (5.54)$$

In this case the decision rule is the following



**Definition 8 (Minimum-Distance Rule)** *To classify a pattern  $\mathbf{x}$  compute the Euclidean distance between  $\mathbf{x}$  and each of the  $\boldsymbol{\mu}_i$  mean vectors and assign the pattern to the class whose mean is the closest.*

A classifier that implements such rule is called *minimum-distance classifier*. The mean vector (or *centroid*)  $\boldsymbol{\mu}_i$  is also viewed as a prototype for a pattern belonging to the class  $\mathcal{C}_i$ .

### Prior Probabilities Are Not All Equal

If the prior probabilities are not all the same, the decision is influenced in favor of the class with the highest a priori probability. In particular, if a pattern  $\mathbf{x}$  has the same distance from two or more different mean vectors, the decision rule chooses the class  $\mathcal{C}_i$  that has the highest *a priori* probability.

Now we consider again the (5.52), it can be rewritten in the following way:

$$\gamma_i(\mathbf{x}) = -\frac{1}{2\sigma^2}[\|\mathbf{x}\|^2 - 2\boldsymbol{\mu}_i^T \mathbf{x} + \|\boldsymbol{\mu}_i\|^2] + \ln p(\mathcal{C}_i). \quad (5.55)$$

Since the term  $\|\mathbf{x}\|^2$  is the same for all  $i$ , it can be considered an additive constant. Therefore it can be neglected and we can obtain the following linear expression:

$$\gamma_i(\mathbf{x}) = \mathbf{a}_i^T \mathbf{x} + b_i \quad (5.56)$$

where:

$$\mathbf{a}_i = \frac{1}{\sigma^2} \boldsymbol{\mu}_i \quad (5.57)$$

$$b_i = -\frac{1}{2\sigma^2} \|\boldsymbol{\mu}_i\|^2 + \ln p(\mathcal{C}_i) \quad (5.58)$$

$b_i$  is often called the *threshold* or *bias* for the  $i^{\text{th}}$  class. The expression (5.56) is called *linear discriminant function*. A classifier based on linear discriminant function is called a *linear classifier*.

In addition, it is possible to show (See Problem 5.11) that the decision surfaces for a linear classifier are hyperplanes. Given two adjacent decision regions  $\mathcal{D}_i$  and  $\mathcal{D}_j$ , the hyperplane separating two regions is orthogonal to the line that joins the respective means  $\boldsymbol{\mu}_i$  and  $\boldsymbol{\mu}_j$ .

### 5.8.2 Covariance Matrix Is The Same for All Classes

In this subsection we discuss another particular case that occurs when the covariance matrix is the same for all the classes. This corresponds to the situation in which the patterns fall in hyperellipsoidal clusters of equal size. We consider again the Equation (5.49):

$$\gamma_i(\mathbf{x}) = -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \ln p(\mathcal{C}_i) \quad (5.59)$$

and we see that the first two terms are independent of  $i$ . Therefore they can be considered additive constants and then neglected. Hence the previous equation can be rewritten as :

$$\gamma_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) + \ln p(\mathcal{C}_i) \quad (5.60)$$

### Prior Probabilities Are All Equal

If the prior probability  $\mathcal{C}_i$  is the same for each class, it becomes an additive constant that can be neglected and becomes:

$$\gamma_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i). \quad (5.61)$$

This is quite similar at the expression that we get when the features are independent. The unique difference is that the Euclidean distance is replaced with Mahalanobis distance. In similar way we can formulate an analogous decision rule

**Definition 9 (*Minimum Mahalanobis Distance Rule*)** *To classify a pattern  $\mathbf{x}$  compute the Mahalanobis distance between  $\mathbf{x}$  and each of the  $\boldsymbol{\mu}_i$  mean vectors and assign the pattern to the class whose mean is the closest.*

A classifier that implements such rule is called *minimum Mahalanobis distance classifier*.

### Prior Probabilities Are Not All Equal

If the prior probabilities are not all the same, the decision is influenced in favor of the class with the highest a priori probability. In particular, if a pattern  $\mathbf{x}$  has the same distance from two or more different mean vectors, the decision rule choose the class  $\mathcal{C}_i$  that has the largest *a priori* probability.

Now we consider again (5.60) that can be rewritten in the following way:

$$\gamma_i(\mathbf{x}) = -\frac{1}{2}[\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i] + \ln p(\mathcal{C}_i) \quad (5.62)$$

The term  $\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}$  non depends by the index  $i$  and it can be considered an additive constant that can be neglected. Hence the discriminant functions are:

$$\gamma_i(\mathbf{x}) = \mathbf{a}_i^T \mathbf{x} + b_i \quad (5.63)$$

where:

$$\mathbf{a}_i = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \quad (5.64)$$

$$b_i = -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \ln p(\mathcal{C}_i). \quad (5.65)$$

Also in this case the discriminant function are linear. The resulting decision surface between two adjacent decision region  $\mathcal{D}_i$  and  $\mathcal{D}_j$  is again an hyperplane, unlike the case of the features that are statistically independent, are not generally orthogonal to the line that joins the means  $\boldsymbol{\mu}_i$  and  $\boldsymbol{\mu}_j$  (See Problem 5.12).

### 5.8.3 Covariance Matrix Is Not the Same for All Classes

In this subsection we discuss the general case that is the covariance matrix is not the same for all the classes. We consider again the Equation (5.49)

$$\gamma_i(\mathbf{x}) = -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \ln p(\mathcal{C}_i).$$

We notice that the unique term that is an additive constant is  $-\frac{n}{2} \ln 2\pi$ . Dropping it we obtain:

$$\gamma_i(\mathbf{x}) = \mathbf{x}^T \mathbf{S}_i \mathbf{x} + \mathbf{a}_i^T \mathbf{x} + b \quad (5.66)$$

where

$$\mathbf{S}_i = -\frac{1}{2} \boldsymbol{\Sigma}_i^{-1} \quad (5.67)$$

$$\mathbf{a}_i = \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i \quad (5.68)$$

$$b = -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i - \frac{1}{2} \ln |\boldsymbol{\Sigma}_i| + \ln p(\mathcal{C}_i). \quad (5.69)$$

The discriminant functions in this case are nonlinear. In particular, in the binary classifiers the decision surfaces are *hyperquadrics*. The results obtained for the binary classifiers can be extended to the case of more than two classes, fixed that are two classes that share the decision surface.

## 5.9 Receiver Operating Curves

In this section we present a graphical method to represent the performances of a classifier, the *Receiver operating curves* [7]. This representation has its roots in the signal detection theory. We consider a device that has to detect an atomic particle (e.g. an electron). The model of our device is simple: if the particle is present the voltage  $v$  assumes a normal distribution  $\mathcal{N}(v_2, \sigma)$ , otherwise the voltage assumes the same normal distribution with the same variance but with different mean that  $\mathcal{N}(v_1, \sigma)$ . The device decides that the particle is present when the voltage  $v$  exceeds a threshold value  $v^*$ . Unfortunately the users of the device do not know the value of the threshold value. Therefore we need a measure, independent of the threshold value, that expresses the effectiveness of the device to detect electrons. A measure that responds to this criterion is the *discriminability*:

$$\delta = \frac{\|v_2 - v_1\|}{\sigma}. \quad (5.70)$$

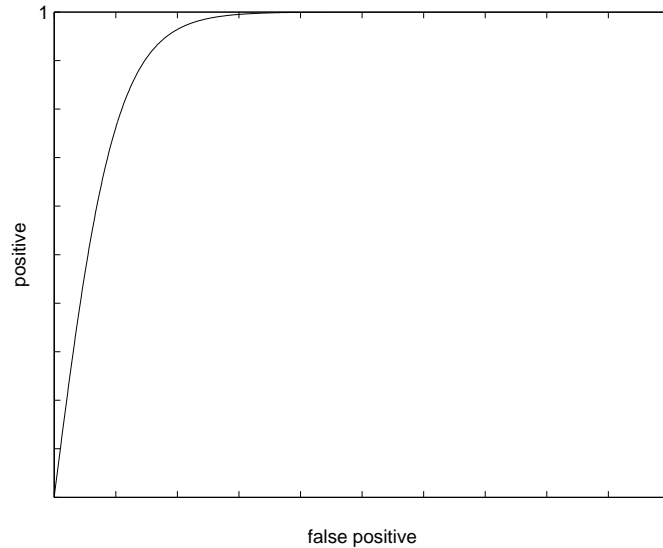
The larger is the discriminability the better is the device.

In general we do not know  $v_1, v_2, \sigma$ , but we know the decisions of the device and we can establish their correctness, for instance using other methods to establish the presence of the particle. We consider the following four probabilities:

- $p(v > v^* | v \in \mathcal{C}_2)$  a *positive* that is the probability that the voltage is higher than  $v^*$  when the particle is present
- $p(v > v^* | v \in \mathcal{C}_1)$  a *false positive* that is the probability that the voltage is higher than  $v^*$  when the particle is absent
- $p(v < v^* | v \in \mathcal{C}_2)$  a *false negative* that is the probability that the voltage is smaller than  $v^*$  when the particle is present
- $p(v < v^* | v \in \mathcal{C}_1)$  a *negative* that is the probability that the voltage is smaller than  $v^*$  when the particle is absent

If we repeat our experiments many times, we can estimate these probabilities experimentally. We can represent our system with a couple of real numbers, namely the positive and the false positive rates. Hence the system can be represented by a point in a two-dimensional space where on x-axis and y-axis are respectively the positive and the false positive rates. If we keep fixed the model only changing the threshold  $v^*$ , the positive and the false positive rates change. In this way the system describes a curve. This curve is called the *receiver operating characteristic* (or *ROC*) curve. The advantage of the signal detection approach consists in distinguishing between *discriminability* and *decision bias*. The discriminability is a specific property of the detection device; the decision bias depends by the receiver.

Each ROC curve is unique, that is, there is one and only one ROC curve that passes through a pair of positive and false positive rates. We can generalize the previous discussion and apply it to two classes having any arbitrary multidimensional distributions. Suppose we have two distributions  $p(\mathbf{x}|\mathcal{C}_1)$  and  $p(\mathbf{x}|\mathcal{C}_2)$  partially overlapped, therefore the Bayes classification error is not null. Any pattern whose state of nature is  $\mathcal{C}_2$  could be correctly classified as  $\mathcal{C}_2$  (a positive in the ROC terminology) or misclassified as  $\mathcal{C}_1$  (a false positive). However, in the multidimensional case we could have many decision surfaces that correspond to different positive rates, each associated with a corresponding false positive rate. In this case a measure of discriminability cannot be determined without knowing the decision rule that yields positive and false positive rates. In addition, we could imagine that the positive and the false positive rates that we have measured are optimal, that is, the decision rule, actually used, is the one that yields the minimum false positive rate. If we build a multidimensional classifier we can represent its performances using a ROC approach. Neglecting the optimality problem, we can simply vary a single parameter in the decision rule and plot the false and negative positive rates. The curve is called the *operating characteristic*. We conclude with the



**Fig. 5.5.** An example of an ROC curve.

remark that the operating characteristic curves are particularly interesting in the applications in which the loss function changes during the time. In this case, if the operating characteristic curve is function of a control parameter, though the loss function changes, it can easily find the value of the control parameter that minimizes the expected risk.

## 5.10 Conclusions

This chapter is a concise description of the foundations of the Bayesian theory of decision. Firstly we have recalled the Bayes theorem and have defined fundamental concepts as likelihood, priors and posterior probability. Then we have defined the Bayes decision rule and have shown its optimality. We have introduced fundamental machine learning concepts such as the loss function and discriminant functions. We have discussed the particular case of Gaussian likelihood deriving the discriminant functions in special case. Finally, we have introduced receiver operating curves.

We conclude the chapter providing some bibliographical remarks. A comprehensive survey of the theory can be found in [7] that covers topics of BDT (e.g. error bounds and Bayesian belief nets) not described by the chapter. Discriminant functions are analyzed in detail in [10]. receiver operating curves are fully discussed in [11]. Finally, a probabilistic approach to the machine learning and decision problem can be found in [6].

## Problems

**5.1.** Given a normal distribution  $\mathcal{N}(\sigma, \mu)$ , show that the percentage of samples that assume values in  $[-3\sigma, 3\sigma]$  exceeds 99%.

**5.2.** Consider the function  $f(x) = \frac{a}{1+x^2}$  where  $a \in \mathbb{R}$ . Find the value  $a$  such that  $f(x)$  is a probability density. Besides, compute the expected value of  $x$ .

**5.3.** Consider the *Geometric distribution*[14] defined by:

$$p(x) = \theta(1 - \theta)^x \quad (x = 0, 1, 2, \dots, 0 \leq \theta \leq 1).$$

Prove that its mean is  $\mathcal{E}[x] = \frac{1-\theta}{\theta}$ .

**5.4.** Given a probability density  $f(x)$ , the *moment of fourth order* [14] is defined by

$$\frac{1}{\sigma^4} \int_{-\infty}^{\infty} f(x)(x - \mu)^4 dx$$

where  $\mu$  and  $\sigma^2$  are, respectively, the mean and the variance.

Prove that the moment of fourth-order of a normal distribution  $\mathcal{N}(\mu, \sigma)$  is 3.

**5.5.** Let  $x = (x_1, \dots, x_\ell)$  and  $y = (y_1, \dots, y_\ell)$  be two variables. Prove that if they are statistically independent their covariance is null.

**5.6.** Suppose we have two classes  $\mathcal{C}_1$  and  $\mathcal{C}_2$  with a priori probabilities  $p(\mathcal{C}_1) = \frac{1}{3}$  and  $p(\mathcal{C}_2) = \frac{2}{3}$ . Suppose that their likelihoods are  $p(x|\mathcal{C}_1) = \mathcal{N}(1, 1)$  and  $p(x|\mathcal{C}_2) = \mathcal{N}(1, 0)$ . Find numerically the value of  $x$  such that the posterior probabilities  $p(\mathcal{C}_1|x)$ ,  $p(\mathcal{C}_2|x)$  are equal.

**5.7.** Suppose we have two classes  $\mathcal{C}_1$  and  $\mathcal{C}_2$  with a priori probabilities  $p(\mathcal{C}_1) = \frac{2}{5}$  and  $p(\mathcal{C}_2) = \frac{3}{5}$ . Suppose that their likelihoods are  $p(x|\mathcal{C}_1) = \mathcal{N}(1, 0)$  and  $p(x|\mathcal{C}_2) = \mathcal{N}(1, 1)$ . Compute the joint probability such that both points  $x_1 = -0.1$ ,  $x_2 = 0.2$  belong to  $\mathcal{C}_1$ .

**5.8.** Suppose we have two classes  $\mathcal{C}_1$  and  $\mathcal{C}_2$  with a priori probabilities  $p(\mathcal{C}_1) = \frac{1}{4}$  and  $p(\mathcal{C}_2) = \frac{3}{4}$ . Suppose that their likelihoods are  $p(x|\mathcal{C}_1) = \mathcal{N}(2, 0)$  and  $p(x|\mathcal{C}_2) = \mathcal{N}(0.5, 1)$ . Compute the likelihood ratio and write the discriminant function.

**5.9.** Suppose we have three classes  $\mathcal{C}_1$ ,  $\mathcal{C}_2$  and  $\mathcal{C}_3$  with a priori probabilities  $p(\mathcal{C}_1) = \frac{1}{6}$ ,  $p(\mathcal{C}_2) = \frac{1}{3}$  and  $p(\mathcal{C}_3) = \frac{1}{2}$ . Suppose that their likelihoods are respectively  $p(x|\mathcal{C}_1) = \mathcal{N}(0.25, 0)$ ,  $p(x|\mathcal{C}_2) = \frac{a}{1+x^2}$  and  $p(x|\mathcal{C}_3) = \frac{1}{b+(x-1)^2}$ . Find the values  $a$  and  $b$  such that likelihoods are density functions and write three discriminant functions.

**5.10.** Implement the whitening transform. Test your implementation transforming *Iris Data* [9], which can be downloaded by <ftp://ics.uci.edu/pub/machine-learning-databases/iris>. Verify that the covariance matrix of the transformed data is the identity matrix.

**5.11.** Suppose that the features are statistically independent and that they have the same variance  $\sigma$ . In this case where the discriminant function is a linear classifier. Given two adjacent decision regions  $\mathcal{D}_1$  and  $\mathcal{D}_2$ , show that their separating hyperplane is orthogonal to the line connecting the means  $\mu_1$  and  $\mu_2$ .

**5.12.** Suppose that the covariance matrix is the same for all the classes. The discriminant function is a linear classifier. Given two adjacent decision regions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  show that their separating hyperplane is *not* orthogonal to the line connecting the means  $\mu_1$  and  $\mu_2$ .





---

## References

1. T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society*, 1763.
2. J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, 1985.
3. J. M. Bernardo and A. F. M. Smith. *Bayesian Theory*. John Wiley, 1986.
4. P. Comon. Independent component analysis: A new concept? *Signal Processing*, 36(1):287–314, 1994.
5. M. H. De Groot. *Optimal Statistical Decisions*. Mc Graw-Hill, 1970.
6. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.
7. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
8. T. S. Ferguson. *Mathematical Statistics: A Decision-Theoretic Approach*. Academic Press, 1967.
9. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
10. K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.
11. D. Green and J.A. Swets. *Signal Detection Theory and Psychophysics*. Wiley, 1974.
12. A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2(1):94–128, 1999.
13. I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.
14. G. A. Korn and T. M. Korn. *Mathematical Handbook for Scientists and Engineers*. Dover, 1961.
15. P. M. Lee. *Bayesian Statistics: An Introduction*. Edward Arnold, 1989.
16. D. V. Lindley. *Making Decisions*. John Wiley, 1991.



## Clustering Methods

### What the reader should know to understand this chapter

- Basic notions of calculus and linear algebra.
- Basic notions of machine learning.
- Programming skills to implement some computer projects proposed in the Problems section.

### What the reader should know after reading this chapter

- The principles of clustering.
- The most popular clustering algorithms.

## 6.1 Introduction

Given a set of examples of a concept, the *learning problem* can be described as finding a general rule that explains examples given only a sample of limited size. Examples are generally referred as *data*. The difficulty of the learning problem is similar to the problem of children learning to speak from the sounds emitted by the grown-up people. The learning problem can be stated as follows: given an example sample of limited size, *to find a concise data description*. Learning methods can be grouped in three big families: *supervised learning*, *reinforcement learning* and *unsupervised learning*.

In supervised learning (or *learning with a teacher*), the data is a sample of input output patterns. In this case, a concise description of the data is the *function* that can yield the output, given the input. This problem is called supervised learning because the objects under considerations are already associated with target values, e.g. classes and real values. Examples of this learning task are the recognition of handwritten letters and digits, the prediction of stock market indexes.

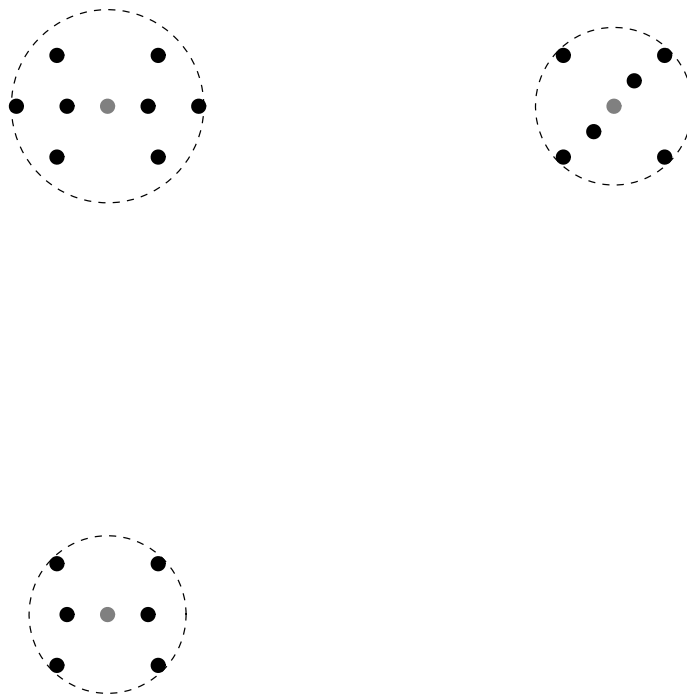
If the data is only a sample of objects without associated target values, the problem is known as unsupervised learning. In unsupervised learning there is no teacher. Hence a concise description of the data could be a set of clusters. Typical examples of unsupervised learning tasks include the problem of image and text segmentation. In unsupervised learning, given a training sample of objects (e.g. images), the aim is to extract some *structure* from them. For instance, identifying indoor or outdoor images or extracting face pixels in an image. If some structure exists in training data, it can take advantage of the redundancy and find a short description of data.

A general way to represent data is to specify a similarity between any pairs of objects. If two objects share much structure, it should be possible to reproduce the data from the same *prototype*. This idea underlies *clustering methods* that form a rich subclass of unsupervised algorithms. It is not possible to provide a formal definition of clustering, only an intuitive definition can be given. Given a fixed number of clusters, we aim to find a grouping of the objects (*clustering*) such that similar objects belong to the same group (*cluster*). If it is possible to find a clustering such that the similarities of the objects in one cluster are much greater than the similarities among objects from different clusters, we have extracted structure from the training sample so that the whole cluster can be represented by one representative data point.

Consider Figure 6.1, the goal of a clustering method, in this case, is to identify the three subsets of black points closely grouped together. Each subset of black points can be represented by one representative data (the grey point). There are some practical reasons for which it is useful to consider clustering methods. In some cases to associate to each sample of the data set the appropriate class (or *label*), as requested by supervised methods, is a time consuming activity. Data sets can contain hundreds of thousand of data, as in the case of handwriting recognition, and some man-months can be required to label the data. Moreover, clustering methods are very useful when the classes are not *a priori* known. For instance, clustering methods can be used in the customer databases of the companies (e.g. insurances, banks, electrical utilities) to individuate groups of customers with the aim of addressing them some marketing actions (e.g. discounts).

Following [18], clustering methods can be categorized into *hierarchical* and *partitioning clustering*. Given a data set to be clustered  $\mathcal{X}$ , hierarchical schemes sequentially build nested clusters with a graphical representation known as *dendrogram*. Partitioning methods directly assign all the data points according to some appropriate criteria, such as similarity and density, into different groups (*clusters*).

In this chapter we focus on the *prototyped-based clustering (PBC)* algorithms, which is the most popular class of partitioning clustering methods. PBC algorithms lead to the identification of a certain number of *prototypes*, i.e. data points that are representative of a cluster, as the grey points in the Figure 6.1. PBC algorithms are so popular that they are often referred simply clustering algorithms.



**Fig. 6.1.** Each cluster of black points can be represented by a representative data, i.e., the gray point.

This chapter presents the most PBC algorithms, paying special attention to neural-based algorithms. The chapter is organized as follows: Section 6.2 reviews the EM algorithm, that is a basic tool of several PBC algorithms; Section 6.3 presents the basic concepts and the common definitions to all clustering algorithms; Section 6.4 describes the algorithm K-Means; Sections 6.5 and 6.6 review some soft competitive learning algorithms, that is, *self-organizing maps*, *neural gas* and *topology representing networks*; general topographic mapping is discussed in Section 6.7. Section 6.8 presents fuzzy clustering algorithms. Section 6.9 reports, for the sake of completeness, a brief description of hierarchical clustering methods. Finally, in Section 6.10 some conclusions are drawn.

## 6.2 Expectation and Maximization Algorithm\*

This section describes the *expectation and maximization* algorithm which is a basic tool of several clustering methods.

Firstly, we recall the definition of the *maximum-likelihood problem*.

We have a density function  $p(\mathbf{x}|\Theta)$  that is governed by the set of parameters  $\Theta$ . We also have a data set  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  and assume that the data vectors of  $\mathcal{X}$  are *i.i.d.*<sup>1</sup> with distribution  $p(\mathbf{x})$ . Therefore, the resulting density for the samples is

$$\mathcal{L}(\Theta|\mathcal{X}) = \prod_{i=1}^{\ell} p(\mathbf{x}_i|\Theta). \quad (6.1)$$

The function  $\mathcal{L}(\Theta|\mathcal{X})$  is called the *likelihood of the parameters given the data*, or simply the *likelihood function*.

The likelihood is thought of as a function of the parameters  $\Theta$  where the data set  $\mathcal{X}$  is fixed.

Now we can state the *maximum likelihood problem*.

**Problem 1** *To find the parameter  $\Theta^*$  that maximizes the likelihood  $\mathcal{L}(\Theta|\mathcal{X})$ , that is,*

$$\Theta^* = \arg \max_{\Theta} \mathcal{L}(\Theta|\mathcal{X}). \quad (6.2)$$

Since the product of several thousands of probabilities is a number too small to be processed with computers, the maximization of the likelihood is addressed through the equivalent maximization of the *loglikelihood*:

$$\Theta^* = \arg \max_{\Theta} \mathcal{L}(\Theta|\mathcal{X}) = \arg \max_{\Theta} \sum_{i=1}^{\ell} \log[p(\mathbf{x}_i|\Theta)]. \quad (6.3)$$

In principle  $\Theta^*$  can be found as the point where the derivative of the loglikelihood with respect to  $\Theta$  is null, but this rarely leads to analytically tractable equations. It is thus necessary to use other techniques for the maximum likelihood estimation of the parameters. The rest of this section introduces the *expectation-maximization* method which is one of the main approaches used to solve such problem.

### 6.2.1 Basic EM\*

The *expectation-maximization* (EM) [3][8][37] algorithm is a general method for finding the Maximum-Likelihood estimate of the parameters of an underlying probability data distribution from a given data set when the data is *incomplete* or the data has missing values. We say that the data is incomplete when not all the necessary information is available. A data set has missing values when there are components of any sample  $\mathbf{x}_i$  whose values are unknown.

There are two main applications of the EM algorithm. The former occurs when the data indeed has missing values, due to limitations of the observation process. The latter occurs when optimizing the likelihood function is analytically intractable and the likelihood function can be simplified by assuming the

---

<sup>1</sup> independent identically distributed.

existence of additional *missing* (or *hidden*) parameters. The latter application is commonly used in clustering.

We assume that the data set  $\mathcal{X}$  is generated by some unknown probability distribution  $p(\mathbf{x})$ . We call  $\mathcal{X}$  the *incomplete data*. We assume that a complete data set  $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$  exists and the joint probability density function is:

$$p(z|\Theta) = p(\mathbf{x}, \mathbf{y}|\Theta) = p(\mathbf{y}|\mathbf{x}, \Theta)p(\mathbf{x}|\Theta).$$

With this new density function, we can define a new likelihood function:

$$\mathcal{L}(\Theta|\mathcal{Z}) = \mathcal{L}(\Theta|\mathcal{X}, \mathcal{Y}) = p(\mathcal{X}, \mathcal{Y}|\Theta) \quad (6.4)$$

called the *complete data likelihood*.

The value of this function can be modeled as a random variable distributed following a unknown density function  $h_{\mathcal{X}, \Theta}(\mathcal{Y})$  where  $\mathcal{X}$  and  $\Theta$  are constants,  $\mathcal{Y}$  is a random variable. The original likelihood function  $\mathcal{L}(\Theta|\mathcal{X})$  is called *incomplete data likelihood function*.

The EM algorithm iteratively performs two steps called *Expectation* and *Maximization*. At each iteration  $i$ , the result is an estimate  $\Theta^{(i)}$  of the parameters. The first estimate  $\Theta^{(0)}$  is usually obtained through a random initialization. After each iteration, the likelihood  $\mathcal{L}^{(i)} = \mathcal{L}(\Theta^{(i)}|\mathcal{X})$  can be estimated. The two steps of the EM algorithm are repeated until the algorithm converges, i.e. until the estimate  $\Theta^{(i)}$  does not change anymore. Each iteration is guaranteed to increase the loglikelihood and the algorithm is guaranteed to converge to a local maximum of the likelihood function.

### E-step\*

The name of this step is due to the fact that is aimed at the estimation of the complete data log likelihood  $\log p(\mathcal{X}, \mathcal{Y}|\Theta)$  with respect to the unknown data  $\mathcal{Y}$  given the observed data  $\mathcal{X}$  and the current parameter estimates  $\Theta^{(i-1)}$ .

We define

$$Q(\Theta, \Theta^{(i-1)}) = \mathcal{E} \left[ \log p(\mathcal{X}, \mathcal{Y}|\Theta) | \mathcal{X}, \Theta^{(i-1)} \right] \quad (6.5)$$

where  $\Theta^{(i-1)}$  are the current parameter estimates,  $\mathcal{E}[\cdot]$  is the expectation operator and  $\Theta$  are the new parameters that we set to maximize  $Q$ .

While  $\mathcal{X}$  and  $\Theta^{(i-1)}$  are constants,  $\Theta$  is the variable to be estimated and  $\mathcal{Y}$  is a random variable governed by the distribution  $f(\mathbf{y}|\mathcal{X}, \Theta^{(i-1)})$ .

The right side of (6.5) can be rewritten as:

$$\mathcal{E}[\log p(\mathcal{X}, \mathcal{Y}|\Theta) | \mathcal{X}, \Theta^{(i-1)}] = \int_{\mathbf{y} \in \mathcal{Y}} \log p(\mathcal{X}, \mathbf{y}|\Theta) f(\mathbf{y}|\mathcal{X}, \Theta^{(i-1)}) d\mathbf{y} \quad (6.6)$$

where  $\mathcal{Y}$  is the range of  $\mathbf{y}$ .

The expression of  $f(\cdot)$  depends on the problem. Where  $f(\cdot)$  has an analytical expression, the problem is simplified.

The evaluation of the equation (6.6) is called the *E-step* of the algorithm.

**M-step\***

The second step (the *M-step*) of the EM algorithm is aimed at finding the parameter set  $\Theta^{(i)}$  *maximizing*  $Q(\Theta, \Theta^{(i-1)})$  (hence the name maximization):

$$\Theta^{(i)} = \arg \max_{\Theta} Q(\Theta, \Theta^{(i-1)}). \quad (6.7)$$

As anticipated, the steps of the EM algorithm are repeated until the algorithm converges. Each iteration is guaranteed to increase the loglikelihood and the algorithm is guaranteed to converge a local maximum of the likelihood function. Many papers (e.g. [8][33][37]) have been dedicated to the convergence rate of EM algorithm, in practice the algorithm converges after few iterations. This is the main reason of the popularity of the EM algorithm in the machine learning community.

**6.3 Basic Notions and Terminology**

This section presents the main notions related to the clustering problem and introduces definitions and terminology used in the rest of the chapter.

**6.3.1 Codebooks and Codevectors**

Let  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  be a data set, where  $\mathbf{x}_i \in \mathbb{R}^n$ . We call *codebook* the set  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  where each element (called *codevector*)  $\mathbf{w}_c \in \mathbb{R}^n$  and  $K \ll \ell$ .

The *Voronoi region* ( $R_c$ ) of the codevector  $\mathbf{w}_c$  is the set of all vectors in  $\mathbb{R}^n$  for which  $\mathbf{w}_c$  is the *nearest vector* (*winner*)

$$R_c = \{\mathbf{x} \in \mathbb{R}^n \mid c = \arg \min_j \|\mathbf{x} - \mathbf{w}_j\|\}.$$

Each Voronoi region  $R_i$  is a *convex polytope*<sup>2</sup> (in some cases unbounded), where the convexity implies that

$$(\forall \mathbf{x}_1, \mathbf{x}_2 \in R_i) \Rightarrow \mathbf{x}_1 + \alpha(\mathbf{x}_2 - \mathbf{x}_1) \in R_i \quad (0 \leq \alpha \leq 1)$$

is fulfilled.

The *Voronoi Set* ( $V_c$ ) of the codevector  $\mathbf{w}_c$  is the set of all vectors in  $\mathcal{X}$  for which  $\mathbf{w}_c$  is the *nearest codevector*

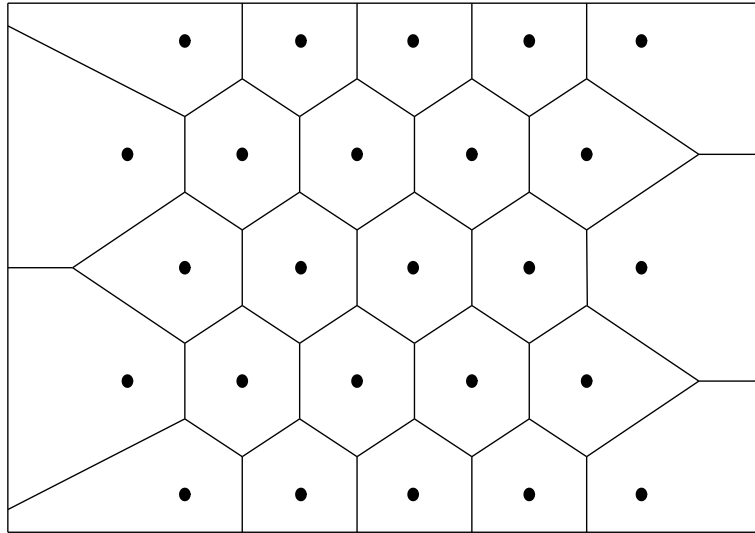
$$V_c = \{\mathbf{x} \in \mathcal{X} \mid c = \arg \min_j \|\mathbf{x} - \mathbf{w}_j\|\}.$$

In Figure 6.2, the Voronoi sets are indicated by the dotted polygons. Voronoi regions and sets are strictly related: suppose that a new input  $\mathbf{x}$  arrives and falls in the Voronoi region of the codevector  $\mathbf{w}$ , this implies that  $\mathbf{x}$  will belong





**Fig. 6.2.** The clusters, formed by the black points, can be represented by their codevectors (grey points). Dashed polygons identify the Voronoi sets associated with each codevector.



**Fig. 6.3.** Codevectors (black points) induce a tessellation of the input space.

to the Voronoi set of the codevector  $\mathbf{w}$ . The partition of  $\mathbb{R}^n$  formed by all Voronoi polygons is called *Voronoi tessellation* (or *Dirichlet tessellation*). An example of Voronoi tessellation is shown in Figure 6.3. Efficient algorithms to compute Voronoi Tessellation are only known for two-dimensional data sets [30][32].

If one connects all pairs of codevectors for which the respective Voronoi regions share an edge, i.e. an  $(n - 1)$ -dimensional hyperface for spaces of dimension  $n$ , one gets the *Delaunay Triangulation*.

<sup>2</sup> In mathematics, polytope is the generalization to any dimension of polygon in two dimensions, polyhedron in three dimensions and polychoron in four dimensions.

### 6.3.2 Quantization Error Minimization

The codebooks are obtained by means of clustering methods. Codebooks are expected to be representative of the data from which they are obtained. A common strategy adopted by clustering methods to obtain a representative codebook consists in the minimization of the *expected quantization error* (or *expected distortion error*). In the case of a continuous input distribution  $p(\mathbf{x})$ , the Expected Quantization Error  $E(p(\mathbf{x}))$  is:

$$E(p(\mathbf{x})) = \sum_{c=1}^K \int_{R_c} \|\mathbf{x} - \mathbf{w}_c\|^2 p(\mathbf{x}) d\mathbf{x} \quad (6.8)$$

where  $R_c$  is the Voronoi region of the codevector  $\mathbf{w}_c$  and  $K$  is the cardinality of the codebook  $W$ .

In the real world we cope with finite data set  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$ . Therefore the minimization of the expected quantization error is replaced with the minimization of the *empirical quantization error*  $E(\mathcal{X})$ , that is:

$$E(\mathcal{X}) = \frac{1}{2\ell} \sum_{c=1}^K \sum_{\mathbf{x} \in V_c} \|\mathbf{x} - \mathbf{w}_c\|^2 \quad (6.9)$$

where  $V_c$  is the Voronoi set of the codevector  $\mathbf{w}_c$ .

When we pass from expected to empirical quantization error, the Voronoi region has to be replaced with the Voronoi set of the codevector  $\mathbf{w}_c$ . A typical application of the empirical quantization error minimization is the *vector quantization* [15][23] (see Section 8.8).

### 6.3.3 Entropy Maximization

An alternative strategy to the quantization error minimization is the entropy maximization. The aim of the entropy maximization is to obtain that the Voronoi set of each codevector roughly has the same number of data. If  $P(s(\mathbf{x}) = \mathbf{w}_c)$  is the probability of  $\mathbf{w}_c$  being the closest codevector for a randomly chosen input  $\mathbf{x}$ , then:

$$P(s(\mathbf{x}) = \mathbf{w}_c) = \frac{1}{K} \quad \forall \mathbf{w}_c \in W \quad (6.10)$$

where  $K$  is the cardinality of the codebook.

If we view the choice of an input  $\mathbf{x}$  and the respective winner codevector  $s(\mathbf{x})$  as a random experiment which assigns a value  $\mathbf{x} \in \mathcal{X}$  to the random variable  $X$ , then (6.10) is equivalent to maximizing the entropy

$$H(X) = - \sum_{\mathbf{x} \in \mathcal{X}} P(\mathbf{x}) \log(P(\mathbf{x})) = \mathcal{E} \left[ \log \left( \frac{1}{P(\mathbf{x})} \right) \right] \quad (6.11)$$

where  $\mathcal{E}[\cdot]$  is the expectation operator.

If the data can be modeled from a continuous probability distribution  $p(\mathbf{x})$ , then (6.10) is equivalent to

$$\int_{R_c} p(\mathbf{x}) d\mathbf{x} = \frac{1}{K} \quad (\forall \mathbf{w}_c \in W) \quad (6.12)$$

where  $R_c$  is the Voronoi region of  $\mathbf{w}_c$  and  $K$  is the cardinality of  $W$ .

When the data set  $\mathcal{X}$  is finite, the equation (6.10) corresponds to the situation where each Voronoi set  $V_c$  contains the same number of data points:

$$\frac{|V_c|}{|\mathcal{X}|} \approx \frac{1}{K} \quad (\forall \mathbf{w}_c \in W). \quad (6.13)$$

An advantage of choosing codevectors to maximize entropy is the inherent robustness of the resulting codebook. The removal of any codevectors affects only a limited fraction of the data.

In general, entropy maximization and quantization error minimization are *antinomic*, i.e. the maximization of the entropy does not lead to the minimization of the quantization error and viceversa. For instance, consider a data set where half of the samples lie in a very small region of the input space, whereas the rest of data are uniformly distributed in the input space. By minimizing the quantization error only one single codevector should be positioned in the pointwise region while all others should be uniformly distributed in the input space. By maximizing entropy half of the codevectors should be positioned in each region.

### 6.3.4 Vector Quantization

An application of the minimization of the empirical quantization error is the *vector quantization* (VQ). The goal of VQ is to replace the data set with the codebook and it has been developed fifty years ago to optimize the transmission over limited bandwidth communication. If the codebook is known by both to sender and receiver, it is adequate to transmit codevector indexes instead of vectors. Therefore, the receiver can use the transmitted index to retrieve the corresponding codevector.

More formally, VQ is the mapping of continuous vectors  $\mathbf{x}$  into a finite set of symbols  $V = \{v_1, \dots, v_K\}$ . Extensive surveys on VQ can be found in [15][26], this section will focus on the general aspects of the VQ problem.

In mathematical terms, a quantizer is composed of two elements. The first is the *encoder*  $\gamma(\mathbf{x})$ :

$$\gamma(\mathbf{x}) : \mathcal{X} \rightarrow V \quad (6.14)$$

which maps  $d$ -dimensional input vectors  $\mathbf{x} \in \mathcal{X}$  into *channel symbols*, i.e. elements of the finite and discrete set  $V$  (see above). The goal of the encoder is to *represent* the data with a set of symbols that require as less space as

possible for transmission and storage purposes. The second is the *decoder*  $\beta(v)$  which maps channel symbols into elements of the reproduction alphabet  $W$ :

$$\beta(v) : V \rightarrow W \quad (6.15)$$

where  $W = (w_1, \dots, w_K)$  is a subset of the input space  $\mathcal{X}$ , i.e. the codebook previously introduced. The goal of the decoder is to *reconstruct* the original data after they have been transmitted or stored as channel symbols. If  $V$  contains  $K$  elements, then  $R = \log_2 K$  is called *rate* of the quantizer, and  $r = R/d$  is called *rate per symbol*.  $R$  corresponds to the minimum number of bits necessary to account for all channel symbols, and  $r$  normalizes such a quantity with respect to the dimensionality of the input space  $\mathcal{X}$ . In general, the quantization is a *lossy* process, i.e. the result  $\hat{\mathbf{x}}$  of the reconstruction is different from the original input  $\mathbf{x}$ . The cost associated to the difference between  $\mathbf{x}$  and  $\hat{\mathbf{x}}$  is called *distorsion* (see below for more details).

In principle, the channel symbols set  $V$  could contain a single element  $v$  and, as a result,  $K = 1$  and  $R = 0$ , i.e. no space is needed for the data. On the other hand, the reduction of  $R$  is constrained by the application needs and the output of the decoder  $\beta(v)$  must satisfy both subjective and objective criteria that account for the quantization quality. The value of  $R$  is then a trade-off between two conflicting needs: the reduction of the number of bits necessary to describe the symbols of  $V$  and the limitation of the distortion. Chapter 2 shows that, in the case of audio quantization, the criteria are signal-to-noise ratio and mean opinion score (MOS), two measures that are particularly suitable for the audio case. In more general terms, the quantization quality can be assessed through the *distortion*, i.e. a cost  $d(\mathbf{x}, \hat{\mathbf{x}})$  associated to the replacement of an input vector  $\mathbf{x}$  with the quantization result  $\hat{\mathbf{x}} = \gamma(\beta(\mathbf{x}))$ .

A quantizer can be considered good when the average distortion:

$$E[d(\mathbf{x}, \hat{\mathbf{x}})] = \lim_{\ell \rightarrow \infty} \frac{1}{\ell} \sum_{i=1}^{\ell} d(\mathbf{x}_i, \hat{\mathbf{x}}_i) \quad (6.16)$$

is low. Such an expression can be applied in practice only when the distribution of  $\mathbf{x}$  is known. However, this is not often the case and the only possible solution is to measure the *empirical* average distortion over a data set of size  $\ell$  sufficiently large to be representative of all possible data:

$$\hat{E}[d(\mathbf{x}, \hat{\mathbf{x}})] = \frac{1}{\ell} \sum_{i=1}^{\ell} d(\mathbf{x}_i, \hat{\mathbf{x}}_i). \quad (6.17)$$

The most common expression of  $d(\mathbf{x}, \hat{\mathbf{x}})$  is the squared error:

$$d(\mathbf{x}, \hat{\mathbf{x}}) = (\mathbf{x} - \hat{\mathbf{x}})^2 \quad (6.18)$$

but other measures can be used [15]. Note that the signal-to-noise ratio expression of Equation (2.30) can be written as follows:

$$SNR = 10 \log_{10} \left\{ \frac{E[\mathbf{x}^2]}{E[d(\mathbf{x}, \hat{\mathbf{x}})]} \right\} \quad (6.19)$$

and it corresponds to the empirical average distortion normalized with respect to the average energy. This enables one to account for the fact that higher distortions can be tolerated at higher energies. On the other hand, the meaning of  $\mathbf{x}^2$  is not necessarily evident when passing from signals characterized by an actual energy (like audio waves) to generic vectors.

A quantizer is said to be *optimal* when it minimizes the average distortion and there are two properties that must be satisfied for a quantizer being optimal [23].

**Definition 10** *Given a specific decoder  $\beta(v)$ , the optimal encoder  $\gamma(\mathbf{x})$  selects the channel symbol  $v^*$  such that:*

$$v^* = \arg \min_{v \in V} d(\mathbf{x}, \beta(v)). \quad (6.20)$$

Since  $v = \gamma(\mathbf{x})$ , the above property means that, given the decoder  $\beta(v)$ , the optimal encoder  $\gamma^*(\mathbf{x})$  is the one performing a *nearest neighbor* mapping:

$$\gamma^*(\mathbf{x}) = \arg \min_{\gamma \in \Gamma} d(\mathbf{x}, \beta(\gamma(\mathbf{x}))), \quad (6.21)$$

where  $\Gamma$  is the set of all possible encoders.

**Definition 11** *Given a specific encoder  $\gamma(\mathbf{x})$ , the optimal decoder  $\beta^*(v)$  assigns each channel symbol  $v$  the centroid of all input vectors mapped into  $v$  by  $\gamma$ :*

$$\beta^*(v) = \frac{1}{N(v)} \sum_{\mathbf{x}_i: \beta(\mathbf{x}_i)=v} \mathbf{x}_i \quad (6.22)$$

where  $N(v)$  is the number of input vectors mapped into  $v$ .

The two properties enable one to obtain a pair  $(\gamma(\mathbf{x}), \beta(v))$  which minimizes the empirical average distortion on a given training set. Note that the clustering algorithms (see Chapter 6) can be interpreted as quantizers. In fact, during the clustering each sample is attributed to a cluster  $v$  and this can be thought of as an encoding operation. Vice versa, each sample can be replaced with the representative of the cluster it belongs to and this can be interpreted as a decoding operation. Moreover, the *empirical quantization error* introduced in Section 6.3 corresponds to the empirical average distortion described above.

## 6.4 K-Means

In this section we will describe the most popular clustering algorithm, *K-Means*. K-Means has two different versions: *batch* and *online*. K-Means. The term batch means at each step the algorithm takes into account the whole data set to update the codebook. Vice versa the term online algorithm indicates that the codebook update is performed after the presentation of each input.

### 6.4.1 Batch K-Means

Batch K-Means [12][24] is the simplest and the oldest clustering method. Despite its simplicity it has been shown to be effective in several applications. Batch K-Means assumes in the literature other names, e.g. in speech recognition is called *Linde-Buzo-Gray (LBG)* algorithm [23], in the old books of pattern recognition is also called *generalized Lloyd algorithm*.

Given a finite data set  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$ , Batch K-Means works by repeatedly moving all codevectors to the arithmetic mean of their Voronoi sets. The theoretical foundation of this procedure is that a *necessary* condition for a codebook  $W$  to minimize the empirical quantization error

$$E(\mathcal{X}) = \frac{1}{2\ell} \sum_{c=1}^K \sum_{\mathbf{x} \in V_c} \|\mathbf{x} - \mathbf{w}_c\|^2$$

is that each codevector  $\mathbf{w}_c$  fulfills the *centroid condition* [16]. In the case of finite data set  $\mathcal{X}$  and the Euclidean distance, the centroid condition reduces to:

$$\mathbf{w}_c = \frac{1}{|V_c|} \sum_{\mathbf{x} \in V_c} \mathbf{x} \quad (6.23)$$

where  $V_c$  is the Voronoi set of the codevector  $\mathbf{w}_c$ .

The batch K-Means algorithm is formed by the following steps:

1. Initialize the codebook  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  with vectors chosen *randomly* from the training set  $\mathcal{X}$ .
2. Compute for each codevector  $\mathbf{w}_i \in W$  its Voronoi Set  $V_i$
3. Move each codevector  $\mathbf{w}_i$  to the mean of its Voronoi Set.

$$\mathbf{w}_i = \frac{1}{|V_i|} \sum_{\mathbf{x} \in V_i} \mathbf{x} \quad (6.24)$$

4. Go to step 2 if any codevector, in the step 3,  $\mathbf{w}_i$  has been changed.
5. Return the codebook.

The second and third steps form a *Lloyd iteration*. It is guaranteed that after a Lloyd iteration the empirical quantization error does not increase. Besides, Batch K-Means can be viewed as an *EM* algorithm (see Section 2). Second and third step are respectively the estimation and the maximization stage. This is important since it means that K-Means is guaranteed to converge after a certain number of iterations.

The main drawback of K-Means is its sensitivity with respect to *outliers*. We recall that outliers are isolated data points whose position in the input space is very far from the remaining data points of the data set. In equation (6.24), we observe that outliers can affect the mean value in the codevector computation. Hence outlier presence can influence significantly codevector positions.

### 6.4.2 Online K-Means

The batch version of the K-Means takes into account the whole data set  $\mathcal{X}$  to update the codebook. When the cardinality of the data set is huge (e.g. several hundreds of thousand of samples), the batch methods are computationally expensive. This can create problems for the storage in the memory or it can take too much time. In this cases the *online update* becomes a necessity.

Online K-Means can be described as follows:

1. Initialize the codebook  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  with vectors chosen *randomly* from the training set  $\mathcal{X}$ .
2. Choose randomly an input  $\mathbf{x}$  according to the input probability function  $p(\mathbf{x})$ .
3. Fix the nearest codevector, i.e the winner  $\mathbf{w}_s = s(\mathbf{x})$

$$s(\mathbf{x}) = \arg \min_{\mathbf{w}_c \in W} \|\mathbf{x} - \mathbf{w}_c\| \quad (6.25)$$

4. Adapt the winner towards  $\mathbf{x}$ :

$$\Delta \mathbf{w}_s = \epsilon(\mathbf{x} - \mathbf{w}_s) \quad (6.26)$$

5. Go to step 2 until a predefined number of iterations is reached.

The fact that only the winner  $s(\mathbf{x})$  is modified for a given input  $\mathbf{x}$  is called *hard competitive learning* or *winner-takes-all* (WTA) learning .

### Winner-Takes-All Learning

A general problem occurring with winner-takes-all learning is the possible existence of *dead codevectors*, i.e. codevectors with an empty Voronoi set. These are codevectors which are never winner for any input and their position never changes. A common way to avoid dead codevectors is to initialize the codevectors according to the sample distribution of the data set. However if the codevectors are initialized randomly according to the input distribution probability  $p(\mathbf{x})$ , then their expected initial local density is proportional to  $p(\mathbf{x})$ . This may be unoptimal if the goal is the quantization error minimization and  $p(\mathbf{x})$  is highly nonuniform. In this case it is better to undersample the region with high probability density, i.e. to use less codevectors than suggested by  $p(\mathbf{x})$ , and to oversample the other regions.

Another drawback of winner-takes-all learning is that different random initializations can yield very different results. For certain initializations, WTA learning may not be able to get the system out of the poor local minimum where it was fallen. One way to cope with this problem is to modify the winner-takes-all learning in a *soft competitive learning*. In this case not only the winner but also some other codevectors are adapted.

### Learning Rate

The online K-Means learning rule, expressed by Equation (6.26), can be justified in the following way. If we compute the derivative of the empirical quantization error  $E(\mathcal{X})$  with respect to the codevector  $\mathbf{w}_s$ , we have:

$$\frac{\partial E(\mathcal{X})}{\partial \mathbf{w}_s} = (\mathbf{x} - \mathbf{w}_s). \quad (6.27)$$

The above equation shows that online K-Means tries to minimize the empirical quantization error using a *steepest gradient descent algorithm* [3]. The *learning rate*  $\epsilon$ , that usually assumes a value between 0 and 1, determines how much the winner is adapted towards the input.

To study how the learning rate value affects the codebook, we observe that at each iteration is modified only the winner codebook. Therefore, for each codevector we consider only the iteration for which it is the winner. To this purpose, we assign to each codevector a time  $t$  that is increased by one only in the iteration, in which the codevector is the winner. Therefore  $t$  allows to compute the number of inputs for which a given codevector  $w_c$  has been winner in the past. For instance,  $t = 5$  means that there were five inputs for which  $w_c$  was the winner codevector. That being said, if the learning rate is constant, i.e.

$$\epsilon = \epsilon_0 \quad (0 < \epsilon_0 \leq 1)$$

then it can be shown that the value of the codevector, at the time  $t$ ,  $\mathbf{w}_c(t)$  can be expressed as an *exponentially decaying average* of those inputs for which the codevector has been the winner, that is:

$$\mathbf{w}_c(t) = (1 - \epsilon_0)^t \mathbf{w}_c(0) + \epsilon_0 \sum_{i=1}^t (1 - \epsilon_0)^{t-i} \mathbf{x}_i^{(c)} \quad (6.28)$$

where  $\mathbf{x}_i^{(c)}$  is the  $i^{th}$  randomly extracted input vector such that  $s(\mathbf{x}) = \mathbf{w}_c$ . Equation (6.28) shows that the influence of past inputs decays exponentially fast with the number of inputs for which the codevector  $\mathbf{w}_c$  is the winner. The most recent input always determines a fraction  $\epsilon$  of the current value of  $\mathbf{w}_c$ . This has the consequence that the algorithm has no convergence. Even after a large number of inputs, the winner codevector can still be remarkably changed by the current input.

To cope with this problem, it has been proposed to have a learning rate that decreases over the time. In particular it was suggested [25] a learning rate which is inverse proportional to the time  $t$ , i.e.

$$\epsilon(t) = \frac{1}{t}. \quad (6.29)$$

Some authors when quote K-Means refers only to online K-Means with a learning rate such as the one defined in (6.29). The reason is that each codevector is always the exact arithmetic mean of the inputs for which it has been winner in the past. We have:



$$\begin{aligned}
\mathbf{w}_c(0) &= \mathbf{x}_0^c \\
\mathbf{w}_c(1) &= \mathbf{w}_c(0) + \epsilon(1)(\mathbf{x}_1^c - \mathbf{w}_c(0)) = \mathbf{x}_1^c \\
\mathbf{w}_c(2) &= \mathbf{w}_c(1) + \epsilon(2)(\mathbf{x}_2^c - \mathbf{w}_c(1)) = \frac{\mathbf{x}_1^c + \mathbf{x}_2^c}{2} \\
&\dots \\
\mathbf{w}_c(t) &= \mathbf{w}_c(t-1) + \epsilon(t)(\mathbf{x}_t^c - \mathbf{w}_c(t-1)) = \frac{\mathbf{x}_1^c + \mathbf{x}_2^c + \dots + \mathbf{x}_t^c}{t} \quad (6.30)
\end{aligned}$$

The set of inputs  $\mathbf{x}_1^c, \mathbf{x}_2^c, \dots, \mathbf{x}_t^c$  for which a particular codevector  $\mathbf{w}_c$  has been the winner may contain elements which lie outside the current Voronoi region of  $V_c$ . Therefore, although  $\mathbf{w}_c(t)$  represents the arithmetic mean of the inputs it has been winner for, at time  $t$  some of these inputs may well lie in Voronoi regions of other units. Another important point about this algorithm that there is no strict convergence, as is present in batch K-Means, since the sum of the harmonic series has no convergence:

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n \frac{1}{i} = \infty$$

Since the series is divergent, even after a large number of inputs and low values of the learning rates  $\epsilon(t)$  large modifications could happen in the winner codevector. However such large modifications have very small probability and many simulations show that the codebook rather quickly assume values that are not changed notably in the further course of the simulation. It has been shown that online K-Means with a learning rate such as the equation (6.29) [25] converges asymptotically to a configuration where each codevector  $\mathbf{w}_c$  is positioned so that it coincides with the expectation value

$$E(\mathbf{x} | \mathbf{x} \in R_c) = \int_{R_c} \mathbf{x} p(\mathbf{x}) d\mathbf{x} \quad (6.31)$$

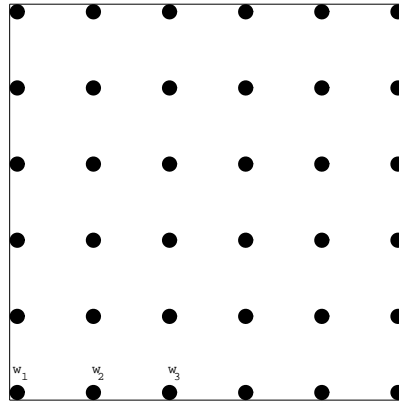
of its Voronoi region  $R_c$ . Equation (6.31) is the generalization, in the continuous case, of the centroid condition (6.23).

Finally another possibility for decaying adaptation rule [34] consists in an exponential decay according to

$$\epsilon(t) = \epsilon_i \left( \frac{\epsilon_f}{\epsilon_i} \right)^{\frac{t}{t_{max}}} \quad (6.32)$$

where  $\epsilon_i$  and  $\epsilon_f$  are the initial and the final values of the learning rate and  $t_{max}$  is the total number of iterations.

The most important drawback of online K-Means is its sensitivity with respect to the input sequence ordering. Changing the order of the input vectors, the algorithm performance can change notably.



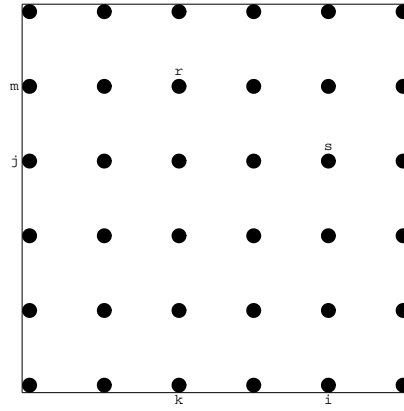
**Fig. 6.4.** In the SOM model, the codevectors are nodes of a two-dimensional grid. For sake of simplicity, only the first three nodes are indicated.

### 6.4.3 K-Means Software Packages

We warmly recommend the reader to implement K-Means as a useful exercise. Nevertheless you can find K-Means software packages in the public-domain *SOM Toolbox* for Matlab 5. The toolbox, developed by Neural Network Research Centre of the University of Helsinki, can be downloaded from <http://www.cis.hut.fi/projects/somtoolbox>.

## 6.5 Self-Organizing Maps

In this section we describe a clustering method, the *self-organizing map* [20][21], which performs a soft competitive learning since other codevectors, in addition to the winner, can be modified. self-organizing map (*SOM*), also called *self-organizing feature map* (*SOFM*) [20], is based on earlier works [35] on the organization of human visual cortex. Although SOM is generally considered a dimensionality reduction method (see Chapter 11), it has been widely used as clustering method. For this reason SOM is included in this chapter. SOM is called a *topology-preserving map* because there is a topological structure imposed on the codevectors. A *topological map* is a mapping that preserves neighborhood relations. In SOM model the topological map consists in a two-dimensional grid  $a_{i,j}$  in which each node is a codevector, as shown in Figure 6.4. The grid is inspired to the *retinotopic map* that connects the retina to the visual cortex in higher vertebrates. For this reason, SOM has *biological plausibility* unlike the other clustering algorithms. We assume, for sake of simplicity, that the grid is rectangular, though other topologies are admitted (e.g. hexagonal) in the model. The grid does not change during self-organization.



**Fig. 6.5.** The distance between the units  $s$  and  $r$  is given by  $d_1(r, s) = |i - k| + |j - m|$ .

The distance on the grid is used to determine how strongly a unit  $r = a_{km}$  is adapted when the unit  $s = a_{ij}$  is the winner.

As shown in Figure 6.5, the metric  $d_1(\cdot)$ , on the grid, is the usual  $L_1$  distance (also called *Manhattan distance*):

$$d_1(r, s) = |i - k| + |j - m| \quad (6.33)$$

The complete SOM algorithm is the following:

1. Initialize the codebook  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  with vectors chosen *randomly* from the training set  $\mathcal{X}$ . Each codevector is mapped onto a unit of the grid. Initialize the parameter  $t$ :

$$t = 0$$

2. Choose randomly an input  $\mathbf{x}$  from the training set  $\mathcal{X}$
3. Determine the winner  $s(\mathbf{x})$ :

$$s(\mathbf{x}) = \arg \min_{\mathbf{w}_c \in W} \|\mathbf{x} - \mathbf{w}_c\| \quad (6.34)$$

4. Adapt each codevector  $\mathbf{w}_r$  according to:

$$\Delta \mathbf{w}_r = \epsilon(t) h(d_1(r, s)) (\mathbf{x} - \mathbf{w}_r) \quad (6.35)$$

where:

$$h(d_1(r, s)) = \exp\left(-\frac{d_1(r, s)^2}{2\sigma(t)^2}\right) \quad (6.36)$$

$$\epsilon(t) = \epsilon_i \left(\frac{\epsilon_f}{\epsilon_i}\right)^{\frac{t}{t_{max}}} \quad (6.37)$$

$$\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{\frac{t}{t_{max}}} \quad (6.38)$$

and  $d_1(r, s)$  is a function that depends on the Manhattan distance between the units  $r$  and  $s$  that are the images of the codevectors  $\mathbf{w}_r$  and  $\mathbf{w}_s$  on the grid.

5. Increase the time parameter  $t$ :

$$t = t + 1 \quad (6.39)$$

6. if  $t < t_{max}$  go to step 2.

It is necessary to remark that the equation (6.36) can be replaced by any decreasing function of the arguments  $\sigma(t)$  and  $d_1(r, s)$ .

### 6.5.1 SOM Software Packages

A public-domain software package, *SOM-PAK* has been developed by T. Kohonen et al. [22]. SOM-PAK, written in C language, can be downloaded from <http://www.cis.hut.fi/research/som-lvq-pak.shtml>. It is also available a *SOM Toolbox* for Matlab 5. The toolbox, developed by Neural Network Research Centre of the University of Helsinki, can be downloaded from <http://www.cis.hut.fi/projects/somtoolbox>. In addition to SOM, SOM Toolbox contains packages for K-Means, principal component analysis [19] and curvilinear component analysis [7].

### 6.5.2 SOM Drawbacks

SOM shares with online K-Means the sensitivity to initialization, the order of input vectors and outliers. Besides, further problems have been identified in [4]:

- The SOM algorithm is not derived by the minimization of a cost function, unlike K-Means that can be obtained by the minimization of the empirical quantization error. Indeed, it has been proved [10] that such a cost function cannot exist for the SOM algorithm.
- Neighborhood-preservation is not guaranteed by the SOM procedure.
- The convergence of SOM algorithm is not guaranteed.

## 6.6 Neural Gas and Topology Representing Network

In this section we describe the *neural gas* and the *topology representing networks*, which do not impose a topology of fixed dimensionality to codevectors. In the case of neural gas there is no topology at all; in the case of topology representing networks the topology of the network depends on the *local dimensionality* of the data and can vary within the input space.

### 6.6.1 Neural Gas

The *neural gas* algorithm [27] sorts for each input  $\mathbf{x}$  the codevectors according to their distance to  $\mathbf{x}$ . The  $n$  codevectors closest to  $\mathbf{x}$  are updated. Hence, neural gas performs a soft competitive learning since other codevectors, in addition to the winner, can be modified. The Neural Gas algorithm is as follows:

1. Initialize the codebook  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  with vectors chosen *randomly* from the training set  $\mathcal{X}$ . Initialize the time parameter  $t$ :

$$t = 0.$$

2. Choose randomly an input  $\mathbf{x}$  from the training set  $\mathcal{X}$
3. Order all elements of  $W$  according to their distance to  $\mathbf{x}$ , i.e. to find the sequence of indices  $(i_0, i_1, \dots, i_{N-1})$  such that  $\mathbf{w}_{i_0}$  is the nearest codevector to  $\mathbf{x}$ ,  $\mathbf{w}_{i_1}$  is the second-closest to  $\mathbf{x}$  and so on. Therefore  $\mathbf{w}_{i_{p-1}}$  is the  $p^{\text{th}}$ -closest to  $\mathbf{x}$ . Following [28] we denote with  $k_i(\mathbf{x}, \mathcal{X})$  the rank number associated with the codevector  $\mathbf{w}_i$ .
4. Adapt the codevectors according to:

$$\Delta \mathbf{w}_i = \epsilon(t) h_{\lambda(t)}(k_i(\mathbf{x}, \mathcal{X})) (\mathbf{x} - \mathbf{w}_i) \quad (6.40)$$

where:

$$\lambda(t) = \lambda_i \left( \frac{\lambda_f}{\lambda_i} \right)^{\frac{t}{t_{max}}} \quad (6.41)$$

$$\epsilon(t) = \epsilon_i \left( \frac{\epsilon_f}{\epsilon_i} \right)^{\frac{t}{t_{max}}} \quad (6.42)$$

$$h_{\lambda(t)}(k_i) = e^{-\frac{k_i}{\lambda(t)}}. \quad (6.43)$$

5. Increase the time parameter  $t$ :

$$t = t + 1 \quad (6.44)$$

6. if  $t < t_{max}$  go to step 2.

### 6.6.2 Topology Representing Network

The main difference with respect to neural gas is that the *topology representing networks (TRN)* [29] model at each adaptation step creates a connection between the winner and the second-nearest codevector. Since the codevectors are adapted according to the neural gas method a mechanism is needed to remove connections which are not valid anymore. This is performed by a local aging connection mechanism. The complete TRN algorithm is the following:

1. Initialize the codebook  $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$  with vectors chosen *randomly* from the training set  $\mathcal{X}$ . Initialize the connection set  $\mathcal{C}$ ,  $\mathcal{C} \subseteq \mathcal{X} \times \mathcal{X}$ , to the empty set  $\mathcal{C} = \emptyset$ . Initialize the time parameter  $t$ :  $t = 0$ .
2. Choose randomly an input  $\mathbf{x}$  from the training set  $\mathcal{X}$ .
3. Order all elements of  $W$  according to their distance to  $\mathbf{x}$ , i.e. to find the sequence of indices  $(i_0, i_1, \dots, i_{K-1})$  such that  $\mathbf{w}_{i_0}$  is the nearest codevector to  $\mathbf{x}$ ,  $\mathbf{w}_{i_1}$  is the second-closest to  $\mathbf{x}$  and so on. Hence  $\mathbf{w}_{i_{p-1}}$  is the  $p^{th}$ -closest to  $\mathbf{x}$ . We denote with  $k_i(\mathbf{x}, \mathcal{X})$  the rank number associated with the codevector  $\mathbf{w}_i$ .
4. Adapt the codevectors according to:

$$\Delta \mathbf{w}_i = \epsilon(t) h_{\lambda(t)}(k_i(\mathbf{x}, \mathcal{X})) (\mathbf{x} - \mathbf{w}_i) \quad (6.45)$$

where:

$$\lambda(t) = \lambda_i \left( \frac{\lambda_f}{\lambda_i} \right)^{\frac{t}{t_{max}}} \quad (6.46)$$

$$\epsilon(t) = \epsilon_i \left( \frac{\epsilon_f}{\epsilon_i} \right)^{\frac{t}{t_{max}}} \quad (6.47)$$

$$h_{\lambda(t)}(k_i) = e^{-\frac{k_i}{\lambda(t)}}. \quad (6.48)$$

5. If it does not exist already, create a connection between  $i_0$  and  $i_1$ :

$$\mathcal{C} = \mathcal{C} \cup \{i_0, i_1\}. \quad (6.49)$$

Set the age of the connection between  $i_0$  and  $i_1$  to zero, *refresh the connection*:

$$age_{(i_0, i_1)} = 0$$

6. Increment the age of all edges emanating from  $i_0$ :

$$age_{(i_0, i)} = age_{(i_0, i)} + 1 \quad (\forall i \in N_{i_0}) \quad (6.50)$$

where  $N_{i_0}$  is the set of direct topological neighbors of the codevector  $w_{i_0}$ .

7. Remove connections with an age larger than maximal age  $T(t)$

$$T(t) = T_i \left( \frac{T_f}{T_i} \right)^{\frac{t}{t_{max}}}. \quad (6.51)$$

8. Increase the time parameter  $t$ :

$$t = t + 1. \quad (6.52)$$

9. If  $t < t_{max}$  go to step 2.

For the time dependent parameters suitable initial values  $(\lambda_i, \epsilon_i, T_i)$  and final values  $(\lambda_f, \epsilon_f, T_f)$  have to be chosen.

Finally we can underline that the cardinality of  $\mathcal{C}$  can be used to estimate the *intrinsic dimensionality*<sup>3</sup> [5] of the data set  $\mathcal{X}$ . See Chapter 11 for more details.

### 6.6.3 Neural Gas and TRN Software Package

A public-domain software package, *GNG*, has been developed by the Institut für Neuroinformatik of Ruhr-Universität of Bochum. GNG can be downloaded from:

*ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/software/NN/DemoGNG.*

The program package, written in Java, contains implementations of Neural Gas and TRN.

### 6.6.4 Neural Gas and TRN Drawbacks

Neural gas and TRN share with other online algorithms (e.g. online K-Means and SOM) the sensitivity to initialization, order of input vectors and outliers. Besides, the convergence of neural gas and TRN is not guaranteed.

## 6.7 General Topographic Mapping\*

In this section we describe *general topographic mapping (GTM)* [4]. Although GTM is generally considered a dimensionality reduction method, it is included in this chapter for its strict connection with SOM. GTM uses an approach different from the clustering methods that we have previously described. GTM does not yield a codebook representative of the data set, but computes an explicit probability density function  $p(\mathbf{x})$  in the data (or input) space. GTM models the probability distribution  $p(\mathbf{x})$  in terms of a number of *latent* (or *hidden*) variables.

### 6.7.1 Latent Variables\*

The goal of a latent variable model is to find a representation for the distribution  $p(\mathbf{x})$  of the data set in an  $N$ -dimensional space in terms of  $L$  latent variables  $\mathbf{X} = (X_1, \dots, X_L)$ . This is achieved by first considering a nonlinear function  $y(\mathbf{X}; W)$ , governed by a set of parameters  $W$ , which maps points  $\mathbf{X}$  in the latent space into corresponding points  $y(\mathbf{X}; W)$  in the input space. We are interested in the situation in which the dimensionality  $L$  of the latent space is lower than the dimensionality  $N$  of the input space, since our

<sup>3</sup> The intrinsic dimensionality of a data set is the minimum number of free variables needed to represent the data without information loss.

premise is that the data itself has an intrinsic dimensionality (see footnote in Section 6.2) which is lower than  $N$ . The transformation  $y(\mathbf{X}, W)$  then maps the latent space into an  $L$ -dimensional *manifold*<sup>4</sup> embedded within the input space. If we define a probability distribution  $p(\mathbf{X})$  on the latent space, this will induce a corresponding distribution  $p(y|W)$  in the input space. We shall refer to  $p(\mathbf{X})$  as the prior distribution of  $\mathbf{X}$ . Since  $L < N$ , the data distribution in input space would be confined to a manifold of dimension  $L$ . Since in reality the data will only approximately lie on a  $L$ -dimensional manifold, it is appropriate to include a noise model for the  $\mathbf{x}$  data vector. We therefore define the distribution of  $\mathbf{x}$ , for given  $\mathbf{X}$  and  $W$ , to be a spherical Gaussian centred on  $y(\mathbf{X}, W)$  having variance  $\sigma^2$  so that  $p(\mathbf{x}|X, W, \sigma^2) \sim \mathcal{N}(\mathbf{x}|y(X, W), \sigma^2\mathbb{I})$ , where  $\mathbb{I}$  is the identity matrix.

The distribution in input space, for a given value of  $W$ , is then obtained by integration over the  $X$ -distribution

$$p(\mathbf{x}|W, \sigma^2) = \int \mathcal{N}(\mathbf{x}|y(\mathbf{X}, W), \sigma^2\mathbb{I})p(\mathbf{X})d\mathbf{X}. \quad (6.53)$$

For a given dataset  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  we can determine the parameter matrix  $W$ , and the variance  $\sigma^2$ , using maximum likelihood principle [9], where the log-likelihood function is given by

$$L(W, \mathcal{X}, \sigma^2) = \sum_{n=1}^{\ell} \log \mathcal{N}(\mathbf{x}_n|y(\mathbf{X}, W), \sigma^2\mathbb{I}). \quad (6.54)$$

In principle we can now seek the maximum likelihood solution for the weight matrix, once we have specified the prior distribution  $p(\mathbf{X})$  and the functional form of the mapping  $y(\mathbf{X}; W)$ , by maximizing  $L(W, \mathcal{X}, \sigma^2)$ .

The latent variable model can be related to the SOM algorithm (see Section 6.5) by choosing  $p(\mathbf{X})$  to be a sum of delta functions centred on the nodes of a regular grid in latent space

$$p(\mathbf{X}) = \frac{1}{K} \sum_{j=1}^K \delta(\mathbf{X} - \mathbf{X}_j),$$

where  $\delta(\cdot)$  is the *Kronecker delta function*.<sup>5</sup> This form of  $p(\mathbf{X})$  allows to compute the integral in (6.53) analytically. Each point  $\mathbf{X}_j$  is then mapped to a corresponding point  $y(\mathbf{X}_j, W)$  in input space, which forms the centre of a Gaussian density function.

Hence the distribution function in input space takes the form of a Gaussian mixture model

<sup>4</sup> We assume, for the sake of simplicity that the definition of a manifold coincides with the one of subspace. The manifold is formally defined in Chapter 11.

<sup>5</sup> The Kronecker delta function  $\delta(x)$  is 1 when  $x = 0$  and 0 otherwise.



$$p(\mathbf{x}|W, \sigma^2) = \frac{1}{K} \sum_{j=1}^K \mathcal{N}(\mathbf{x}|y(\mathbf{X}_j, W), \sigma^2 \mathbb{I})$$

and the log likelihood function (6.54) becomes

$$L(W, \mathcal{X}, \sigma^2) = \sum_{n=1}^{\ell} \log \left[ \frac{1}{K} \sum_{j=1}^K \mathcal{N}(\mathbf{x}_n|y(\mathbf{X}_j, W), \sigma^2 \mathbb{I}) \right]. \quad (6.55)$$

This distribution is a constrained Gaussian mixture since the centers of the Gaussians cannot move independently but are related through the function  $y(\mathbf{X}, W)$ . Since the mapping function  $y(\mathbf{X}, W)$  is smooth and continuous, the projected points  $y(\mathbf{X}_j, W)$  will necessarily have a *topographic ordering* in the sense that any two points  $\mathbf{x}_l$  and  $\mathbf{x}_{l'}$  are close in latent space will map to points  $y(\mathbf{x}_1, W)$   $y(\mathbf{x}_2, W)$ , which are close in the data space.

### 6.7.2 Optimization by EM Algorithm\*

GTM maximizes Equation (6.55) by means of an EM algorithm (see Section 6.2). By making a careful choice of the model  $y(\mathbf{X}, W)$  we will see that the M-step can be solved exactly. In particular we shall choose  $y(\mathbf{X}, W)$  to be given by a generalized linear network model of the form

$$y(\mathbf{X}, W) = W\phi(\mathbf{X}) \quad (6.56)$$

where the elements of  $\phi(\mathbf{X}) = (\phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x}))$  are  $M$  fixed basis functions  $\phi_i(\mathbf{x})$  and  $W$  is a  $N \times M$  matrix with elements  $w_{ki}$ .

By setting the derivatives of (6.55) with respect to  $w_{ki}$  to zero, we obtain

$$\Phi^T G \Phi W^T = \Phi^T R T \quad (6.57)$$

where  $\Phi$  is a  $K \times M$  matrix with elements  $\Phi_{ij} = \phi_i(\mathbf{X}_j)$ ,  $T$  is a  $\ell \times N$  matrix with elements  $x_{kn}$  and  $R$  is a  $K \times \ell$  matrix with elements  $R_{jn}$  given by:

$$R_{jn}(W, \sigma^2) = \frac{\mathcal{N}(\mathbf{x}_n|y(\mathbf{X}_j, W), \sigma^2 \mathbb{I})}{\sum_{s=1}^K \mathcal{N}(\mathbf{x}_n|y(\mathbf{X}_s, W), \sigma^2 \mathbb{I})} \quad (6.58)$$

which represent the posterior probability, or *responsibility*, of the mixture component  $j$  for the data point  $n$ .

Finally,  $G$  is a  $K \times K$  diagonal matrix, with elements  $G_{jj}$

$$G_{jj} = \sum_{n=1}^{\ell} R_{jn}(W, \sigma^2)$$

Equation (6.57) can be solved for  $W$  using standard matrix inversion techniques. Similarly, optimizing with respect to  $\sigma^2$  we obtain

$$\sigma^2 = \frac{1}{\ell N} \sum_{j=1}^K \sum_{n=1}^{\ell} R_{jn}(W, \sigma^2) \|y(\mathbf{X}_j, W) - \mathbf{x}_n\|^2. \quad (6.59)$$

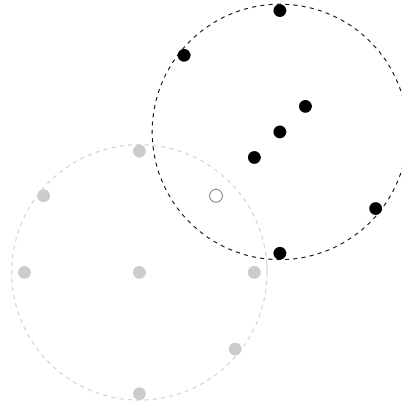
The equation (6.58) corresponds to the E-step, while the equations (6.57) and (6.59) corresponds to the M-step. Hence GTM is convergent. An online version of GTM has been obtained by using the Robbins-Monro procedure to find a zero of the objective function gradient, or by using an online version of the EM algorithm.

### 6.7.3 GTM versus SOM\*

The list below describes some SOM drawbacks and how the GTM algorithm addresses them.

- The SOM algorithm is not derived by optimizing a cost function, unlike GTM.
- In GTM the neighborhood-preserving nature of the mapping is an automatic consequence of the choice of a smooth, continuous function  $y(x, W)$ . Neighbourhood-preservation is not guaranteed by the SOM procedure.
- Convergence of SOM algorithm. Vice versa, convergence of the batch GTM algorithm is guaranteed by the EM algorithm, and the Robbins-Monro theorem provides a convergence proof for the online version.
- GTM defines an explicit probability density function in data space. In contrast, SOM does not define a density model. The advantages of having a density model include the ability to deal with missing data and the straightforward possibility of using a mixture of such models, again trained using EM.
- For SOM the choice of how the neighborhood function should shrink over time during training is arbitrary and so this must be optimized empirically. There is no neighborhood function to select for GTM.
- It is difficult to know by what criteria to compare different runs of the SOM procedure. For GTM one simply compares the likelihood of the data under the model, and standard statistical tests can be used for model comparison.

Nevertheless there are very close similarities between SOM and GTM techniques. At an early stage of the training the responsibility for representing a particular data point is spread over a relatively large region of the map. As the EM algorithm proceeds so this responsibility *bubble* shrinks automatically. The responsibilities (computed in the E-step) govern the updating of  $W$  and  $\sigma^2$  in the M-step and, together with the smoothing effect of the basis functions  $\phi_i(x)$ , play an analogous role to the neighbourhood function in the



**Fig. 6.6.** The two clusters (black points and grey points) are partially overlapped. The circle indicates a point that is assigned to both clusters.

SOM algorithm. While the SOM neighbourhood function is arbitrary, however, the shrinking responsibility bubble in GTM arises directly from the EM algorithm.

#### 6.7.4 GTM Software Package

A *GTM* Toolbox for Matlab, has been developed [4]. The toolbox can be downloaded from <http://www.ncrg.aston.ac.uk/GTM>.

## 6.8 Fuzzy Clustering Algorithms

While in the algorithms described so far, each input  $\mathbf{x}$  belongs to one and only one cluster, in fuzzy clustering algorithms the data points are assigned to several clusters with varying degrees of *membership*.

The idea is based on the observation that, in real data, data clusters usually overlap to some extent and it is difficult to trace clear borders among them. Therefore, some data vectors cannot be certainly assigned to exactly one cluster and it is more reasonable to assign partially to several clusters. Consider the Figure 6.6 the two clusters, formed by the black and the grey points, are partially overlapped. Hence it is reasonable to suppose that some points (e.g. the circle) are assigned to both clusters. This section provides a brief description of the most popular and widely applied fuzzy clustering algorithm, the *fuzzy C-Means algorithm (FCM)* [2]. For comprehensive surveys on fuzzy clustering algorithms, see [1][18].

### 6.8.1 FCM

Let  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  be a data set, where  $\mathbf{x}_i \in \mathbb{R}^n$  and  $W = (\mathbf{w}_1, \dots, \mathbf{w}_C)$  the codebook. As the K-Means algorithm, FCM assumes that the number of clusters is a priori known. Unlike K-Means, the number of clusters is called  $C$ . FCM minimizes the cost function:

$$J_{FCM} = \sum_{i=1}^C \sum_{j=1}^{\ell} u_{ij}^S \|\mathbf{x}_j - \mathbf{w}_i\|^2 \quad (6.60)$$

subject to the  $m$  probabilistic constraints:

$$\sum_{i=1}^C u_{ij} = 1 \quad j = 1 \dots, \ell.$$

Here,  $u_{ij}$  is the membership values of input vector  $\mathbf{x}_j$  belonging to the cluster  $i$ ,  $S$  stands for the degree of fuzziness. Using Lagrangian multipliers method the condition for local minima of  $J_{FCM}$  is derived as

$$u_{ij} = \left[ \sum_{k=1}^C \left[ \frac{\|\mathbf{x}_j - \mathbf{w}_i\|}{\|\mathbf{x}_j - \mathbf{w}_k\|} \right]^{\frac{2}{S-1}} \right]^{-1} \quad \forall i, j \quad (6.61)$$

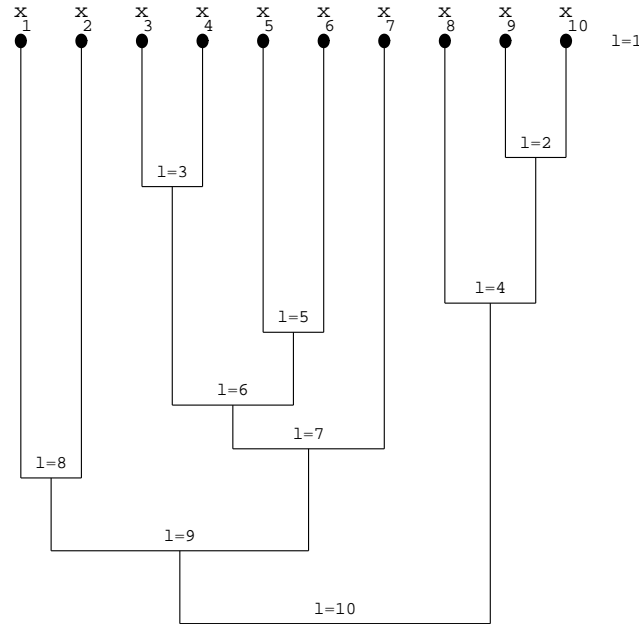
and

$$\mathbf{w}_i = \frac{\sum_{j=1}^{\ell} u_{ij}^S \mathbf{x}_j}{\sum_{j=1}^{\ell} u_{ij}^S} \quad \forall i. \quad (6.62)$$

The final cluster centers can be obtained by the iterative optimization scheme, called the *alternative optimization (AO)* [31] method. The online version for the optimization of  $J_{FCM}$  with stochastic gradient descent method is known as *fuzzy competitive learning* [6].

## 6.9 Hierarchical Clustering

In this section we briefly discuss an alternative clustering approach to the PBC methods previously described in the rest of the chapter, i.e. the *hierarchical clustering*. PBC methods do not assume the existence of substructures in the clusters. Nevertheless, it can happen that data are organized hierarchically, i.e. clusters have subclusters and subclusters have subsubclusters and so on. In this case PBC methods are not effective and have to be replaced with alternative methods, i.e. hierarchical clustering methods. We pass to introduce them. Given a data set  $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^n$ , we consider a sequence of



**Fig. 6.7.** A dendrogram.

partitions of its elements into  $K$  clusters, where  $K \in [1, \ell]$  is an integer not fixed a priori. The first possible partition of  $\mathcal{X}$  is the one into  $\ell$  clusters, where each cluster has a single element. The second partition divides  $\mathcal{X}$  into  $\ell - 1$  clusters and so on until the  $\ell^{\text{th}}$  partition in which all data samples are grouped in a single cluster. The generic  $l^{\text{th}}$  partition, that we simply call the partition at  $l^{\text{th}}$  level, has  $K$  clusters where  $K = \ell - l + 1$ . Given any two data samples  $\mathbf{x}_A$  and  $\mathbf{x}_B$ , at some level they will belong to the same cluster. If the partition sequence is such that whenever two data samples are elements of the same cluster at level  $\alpha$  remain elements of the same cluster at the levels higher than  $\alpha$ , the sequence is called *hierarchical clustering*. The hierarchical clustering is generally represented by means of a tree, called *dendrogram*. A dendrogram for a data set with ten samples is shown in Figure 6.7. At level  $l = 1$  each cluster has a single pattern. At level  $l = 2$ ,  $x_9$  and  $x_{10}$  are gathered in a single cluster. At last level,  $l = 10$ , all pattern belong to a single cluster.

Hierarchical clustering methods can be grouped in two different families: *agglomerative* and *divisive*. Agglomerative methods use a bottom-up approach, i.e. they start with  $\ell$  clusters formed by a single pattern and build the partition sequence merging them successively. Divisive methods are top-down i.e. they start with a single cluster in which the patterns are gathered and at the second level the cluster is splitted in two other clusters and so on. Therefore the partition sequence is built splitting clusters successively. For sake of

simplicity, we only describe the agglomerative methods. The most popular agglomerative method is the so-called *agglomerative hierarchical clustering* (AHC). AHC is formed by the following steps:

1. Given a dataset  $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$ , choose  $K$  and initialize  $\hat{K} = \ell$  and  $\mathcal{S}_i = \{\mathbf{x}_i\}$  ( $i = 1, \dots, \ell$ ).
2.  $\hat{K} = \hat{K} - 1$
3. Find the two *nearest* clusters  $\mathcal{S}_i$  and  $\mathcal{S}_j$
4. Merge  $\mathcal{S}_i$  and  $\mathcal{S}_j$ , i.e.  $\mathcal{S}_i = \mathcal{S}_i \cup \mathcal{S}_j$  and delete  $\mathcal{S}_j$ .
5. If  $\hat{K} \neq K$  go to step 2
6. return  $K$  clusters  $\mathcal{S}_i$

If in the AHC algorithm we choose  $K = 1$  the algorithm produces a single cluster and we obtain a dendrogram like the one described in figure 6.7. The second step of AHC finds among clusters  $\mathcal{S}_i$  the two nearest ones. In order to find the nearest clusters, we need to measure, for each couple of clusters  $\mathcal{S}_A$  and  $\mathcal{S}_B$  their distance. Many definitions of distance between clusters [9] have been proposed, the most popular are:

$$D_{min}(\mathcal{S}_A, \mathcal{S}_B) = \min_{\mathbf{x} \in \mathcal{S}_A; \mathbf{y} \in \mathcal{S}_B} \|\mathbf{x} - \mathbf{y}\| \quad (6.63)$$

$$D_{max}(\mathcal{S}_A, \mathcal{S}_B) = \max_{\mathbf{x} \in \mathcal{S}_A; \mathbf{y} \in \mathcal{S}_B} \|\mathbf{x} - \mathbf{y}\| \quad (6.64)$$

When Equation (6.63) is used to measure the distance between clusters, AHC is referred as *nearest-neighbor cluster algorithm* or *minimum algorithm*. Vice versa, when Equation (6.64) is used AHC is called *farthest-neighbor cluster algorithm* or *maximum algorithm*.

Some variants of the AHC algorithms have been proposed, reader can find further details in [9].

## 6.10 Conclusion

This chapter has presented the most popular and widely applied prototype-based clustering algorithms, with a special attention to neural-based algorithms. Firstly we have recalled the expectation and maximization algorithm, that is the basic tool of several clustering algorithms. Then the chapter has described both batch and online versions of the K-Means algorithm, some competitive learning algorithms (SOM, neural gas and TRN) and the general topographic mapping with a discussion about its connections with SOM. We have described only algorithms whose codevector number has to be fixed *a priori*. Clustering algorithms whose codevector number has not necessarily to be fixed can be found in [13][14]. Clustering methods which produce nonlinear separation surfaces among data, i.e. kernel and spectral clustering methods, will be discussed in Chapter 9.

None of the algorithms described in the chapter is better than the others. On the other hand, the evaluation of a clustering technique is a difficult problem. The clustering leading to the *best* results is assumed to perform better than the others. The concept of the best clustering depends on the application. The best clustering can be the one that minimizes the quantization error but not necessarily. As an example, consider a clustering application which performs a vector quantization to reduce the amount of data, to be transmitted through a channel. In this case, the performance measure of the process is the quality of the signal after the transmission. The use of different clustering methods techniques will result in a different quality of the output signal that provides an indirect measure of the clustering effectiveness. However, the literature offers some directions to assess the clustering algorithm *robustness*.

We call the *assumed model* of a clustering algorithm, the ensembles of the assumptions (e.g. the *model assumptions*) on which the algorithm is based. Examples of the assumptions are the absence of the outliers and data are i.i.d. Following [17] a robust clustering algorithm should possess the following properties:

1. it should have a reasonably good accuracy at the assumed model;
2. small deviations from the model assumption should affect only slightly the performance;
3. larger deviations from the model assumption should not cause a catastrophe, i.e. the algorithm performances decrease dramatically.

The algorithms presented in this chapter satisfy in general the first condition, but often lack in addressing the other issues.

Finally, we conclude the chapter providing some bibliographical remarks. A good survey on clustering methods can be found in [18]. A comprehensive survey of SOM model can be found in [21]. Neural gas and TRN are described in [28][29]. GTM is fully discussed in [4]. Fuzzy clustering methods are widely reviewed in [1]. Hierarchical clustering methods are described in detail in [9].

## Problems

**Problem 6.1.** Implement batch K-Means and test it on *Iris Data* [11] that can be downloaded at <ftp.ics.uci.edu/pub/machine-learning-databases/iris>. Plot the quantization error versus the number of iterations.

**Problem 6.2.** Can K-Means separate clusters nonlinearly separated using only two codevectors? And neural gas and SOM? Explain your answers.

**Problem 6.3.** Study experimentally (e.g. on Iris Data) how the initialization affects K-Means performances.

**Problem 6.4.** Suppose that the *empirical quantization error*  $E(\mathcal{X})$  of a data set  $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  assumes the following form:

$$E(\mathcal{X}) = \frac{1}{2\ell} \sum_{c=1}^K \sum_{x \in V_c} (G(\mathbf{x}, \mathbf{x}) - 2G(\mathbf{x}, \mathbf{w}_c) + G(\mathbf{w}_c, \mathbf{w}_c))$$

where the function  $G(\cdot)$  is  $G(x, y) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2}\right)$ . Find the online K-Means learning rule, in this case.

**Problem 6.5.** Suppose that the *empirical quantization error*  $E(\mathcal{X})$  of a data set  $\mathcal{X}$  assumes the form of Exercise 4. Find the neural gas learning rule.

**Problem 6.6.** Implement K-Means online and test it on *Wisconsin Breast Cancer Database* [36] which can be downloaded at <ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin>. Compare its performances with Batch K-Means's ones. Use in both cases only two codevectors.

**Problem 6.7.** Use SOM-PAK on Wisconsin Breast Cancer Database. Divide the data in three parts. Train SOM on the first part of data (*training set*) changing number of codevectors and other neural network parameters (e.g. learning rate). Select the neural network configuration (*best SOM*) that has the best performance on the second part of data (*validation set*). Finally measure the best SOM performances on the third part of data (*test set*).

**Problem 6.8.** Using the function *sammon* of SOM-PAK visualize the codebook produced by *best SOM* (see Exercise 7).

**Problem 6.9.** Permute randomly Wisconsin Breast Cancer Database and repeat again the Exercise 7. Compare and discuss the results.

**Problem 6.10.** Implement neural gas and test it on *Spam Data* which can be downloaded at <ftp.ics.uci.edu/pub/machine-learning-databases/spam>. Use only two codevectors.



---

## References

1. A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. *IEEE Transactions on System, Man and Cybernetics-B*, 29(6):778–801, 1999.
2. J. C. Bedzek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, 1981.
3. C. M. Bishop. *Neural Networks for Pattern Recognition*. Cambridge University Press, 1995.
4. C. M. Bishop, M. Svensen, and C. K. I. Williams. GTM: the generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.
5. F. Camastra. Data dimensionality estimation methods: A survey. *Pattern Recognition*, 36(12):215–234, 2003.
6. F.L. Chung and T. Lee. Fuzzy competitive learning. *Neural Networks*, 7(3):539–551, 1994.
7. P. Demartines and J. Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping in cluster analysis. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.
8. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal Royal Statistical Society*, 39(1):1–38, 1977.
9. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
10. E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67(1):47–55, 1992.
11. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
12. E. Forgy. Cluster analysis of multivariate data; efficiency vs. interpretability of classifications. *Biometrics*, 21(1):768, 1965.
13. B. Fritzke. Growing cell structures- a self organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.
14. B. Fritzke. A growing neural gas learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.
15. R. Gray. Vector quantization. *IEEE Transactions on Acoustics, Speech and Signal Processing Magazine*, 1(2):4–29, 1984.

16. R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer, 1992.
17. P. J. Huber. *Robust Statistics*. John Wiley, 1981.
18. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surveys*, 31(3):264–323, 1999.
19. I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.
20. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.
21. T. Kohonen. *Self-Organizing Map*. Springer-Verlag, 1997.
22. T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som-pak: The self-organizing map program package. Technical report, Laboratory of Computer and Information Science, Helsinki University of Technology, 1996.
23. Y. Linde, A. Buzo, and R. Gray. Least square quantization in pcm. *IEEE Transaction on Information Theory*, 28(2):129–137, 1982.
24. S. P. Lloyd. An algorithm for vector quantizer design. *IEEE Transaction on Communications*, 28(1):84–95, 1982.
25. J. Mac Queen. Some methods for classifications and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, pages 281–297. University of California Press, 1967.
26. J. Makhoul, S. Roucos, and H. Gish. Vector Quantization in speech coding. *Proceedings of IEEE*, 73(11):1551–1588, 1985.
27. T. E. Martinetz and K. J. Schulten. A "neural gas" network learns topologies. In *Artificial Neural Networks*, pages 397–402. North-Holland, 1991.
28. T. E. Martinetz and K. J. Schulten. Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transaction on Neural Networks*, 4(4):558–569, 1993.
29. T. E. Martinetz and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.
30. S. M. Omohundro. The delaunay triangulation and function learning. Technical report, International Computer Science Institute, 1990.
31. N. R. Pal, K. Pal, and J. C. Bedzek. A mixed c-means clustering model. In *Proceedings of IEEE International Conference on Fuzzy Systems*, pages 11–21. IEEE Press, 1997.
32. F. P. Preparata and M. I. Shamos. *Computational geometry*. Springer-Verlag, 1990.
33. R. Redner and H. Walker. Mixture densities, maximum likelihood and the em algorithm. *SIAM Review*, 26(2), 1984.
34. H. J. Ritter, T. M. Martinetz, and K. J. Schulten. *Neuronale Netze*. Addison-Wesley, 1991.
35. D. J. Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society London*, B194(1117):431–445, 1976.
36. W. H. Wolberg and O. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences, U.S.A.*, 87(1):9193–9196, 1990.
37. C. F. J. Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

## Foundations of Statistical Learning and Model Selection

### What the reader should know to understand this chapter

- Basic notions of machine learning.
- Notions of calculus.
- Chapter 5.

### What the reader should know after reading in this chapter

- Bias-variance dilemma.
- Model selection and assessment.
- Vapnik-Chervonenkis theory.
- Vapnik-Chervonenkis dimension.
- BIC, AIC.
- Minimum description length.
- Crossvalidation.

## 7.1 Introduction

This chapter has two main topics the the *model selection* and the *learning problem*.

Supervised machine learning methods are characterized by the presence of the parameters that have to be tuned to obtain the best performances. The same learning algorithm can be trained using different configurations of parameters generating a different learning machine. The problem of selecting among different learning machines the best one is called *model selection*. We will review the main model selection methods discussing their connections with statistical learning theory.

The learning problem will be discussed under statistical point of view introducing the main issues of *statistical learning theory* (or *Vapnik-Chervonenkis theory*).

The chapter is organized as follows: Section 7.2 describes the bias and variance that are the simplest quantities to measure the performances of a learning machine. The complexity of a learning machine is discussed in Section 7.3. Section 7.4 introduces intuitively the *Vapnik-Chervonenkis dimension* (or *VC dimension*). The main results of the Vapnik-Chervonenkis theory of learning and the formal definition of VC dimension are presented in Section 7.5. Section 7.6 presents two criteria for model selection, i.e. *Bayesian Information Criterion (BIC)* and *Akaike Information Criterion (AIC)*. In Section 7.7 the *minimum description length (MDL)* approach to the model selection is discussed showing that it is equivalent to the BIC criterion; crossvalidation, which is the one of the most popular methods for model selection is reviewed in Section 7.8. Finally, in Section 7.9 some conclusions are drawn.

## 7.2 Bias-Variance Dilemma

In this section we will introduce two new quantities, the *bias* and the *variance*, which can be used to measure the performance of a supervised learning machine. The bias measures the *accuracy* of the learning machine, i.e. how much the output of the learning machine is close to its learning target. Large bias indicates that the output of the machine is not close to its target, that is the learning machine is a *poor learner*.

The variance measures the *precision* of the learning. Large variance indicates that the output of the machine has a large interval of confidence, i.e. the machine is not precise in learning. A learning machine which is not precise in learning is called a *weak learner*. In the rest of the section we will show that bias and variance are not independent. They generate the so-called phenomenon of *bias-variance dilemma*. Firstly, we will discuss the bias and the variance in the case of regression.

### 7.2.1 Bias-Variance Dilemma for Regression

Consider a function  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ . We try to estimate  $F(\cdot)$  using samples of the set  $\mathcal{D}$  that has been generated by  $F(\mathbf{x})$ . We indicate with  $f(\mathbf{x})$ , the estimate of  $F(\mathbf{x})$ . The quality of the estimate can be measured by the mean square error. If we indicate with  $\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x}))^2]$  the average error over all training sets  $\mathcal{D}$  of the same cardinality  $\ell$ , it is possible to show (see Problem 7.1) that it is equal to:

$$\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x}))^2] = (\mathcal{E}[f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x})])^2 + \mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - \mathcal{E}[f(\mathbf{x}, \mathcal{D})])^2]. \quad (7.1)$$

The term  $\mathcal{E}[f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x})]$  is called the *bias*, that is the difference between the expected value and the true value (often not known) of the function. The term  $\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - \mathcal{E}[f(\mathbf{x}, \mathcal{D})])^2]$  is called the *variance*. A small bias means that the estimate of  $F(\cdot)$  has a large accuracy. A small variance indicates that the estimate of  $F(\cdot)$  varies a little changing the training set  $\mathcal{D}$ .

Summing up, the mean-square error can be decomposed as the sum of the square of the bias and the variance. Such decomposition is called the *bias-variance dilemma* or *bias-variance trade-off* [13].

If a learning algorithm, that we call simply a *model*, has many parameters it will be characterized by a low bias, since it usually fits very well the data. At the same time, the model will be characterized by a large variance since it overfits the data.

On the other hand, if the model has a small number of parameters, it will be characterized by a large bias, since it usually does not fit well the data. At the same time, the model will be characterized by a small variance, since the fit does not vary much changing the data set. Finally, we point out that the best strategy consists in keeping low variance and bias at the same time. This strategy can be generally implemented when we have information about the function that has to be approximated.

### 7.2.2 Bias-Variance Decomposition for Classification\*

In this section we discuss the bias-variance decomposition for classification. For sake of simplicity, we only consider the case of binary classification. Let  $\gamma : \mathbb{R}^n \rightarrow \{0, 1\}$  be the discriminant function. If we consider  $\gamma(\cdot)$  under a Bayesian viewpoint, we have:

$$\gamma(\mathbf{x}) = P(y = 1|\mathbf{x}) = 1 - P(y = 0|\mathbf{x}). \quad (7.2)$$

Now we study the binary classification problem using the same approach used for regression. Let  $y(\mathbf{x})$  be a discriminant function (see Chapter 5), defined by:

$$y(\mathbf{x}) = \gamma(\mathbf{x}) + \phi \quad (7.3)$$

where  $\phi$  is a zero-mean random variable having a binomial distribution with variance

$$\sigma^2(\phi|\mathbf{x}) = \gamma(\mathbf{x})(1 - \gamma(\mathbf{x})).$$

The function, that has to be approximated,  $\gamma(\cdot)$  can be represented in the following way:

$$\gamma(\mathbf{x}) = \mathcal{E}(y|\mathbf{x}). \quad (7.4)$$

If we want to apply the same framework of the regression, we have to look for an estimate  $f(\mathbf{x}, \mathcal{D})$  that minimizes the usual mean square error, that is:

$$\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - y)^2]. \quad (7.5)$$

In addition, we assume that the two classes  $\mathcal{C}_1, \mathcal{C}_2$  have the same prior probabilities, that is:

$$P(\mathcal{C}_1) = P(\mathcal{C}_2) = \frac{1}{2}.$$

Therefore the Bayes discriminant has threshold  $y_b = \frac{1}{2}$  and yields a decision boundary formed by patterns such that  $\gamma(\mathbf{x}) = \frac{1}{2}$ .

Given a training set  $\mathcal{D}$  if the classification error is equal to the error of the Bayes discriminant, it assumes the smallest error (*Bayes discriminant error*), that is:

$$P(f(\mathbf{x}, \mathcal{D}) = y) = P(y_b(\mathbf{x}) \neq y) = \min[\gamma(\mathbf{x}), 1 - \gamma(\mathbf{x})]. \tag{7.6}$$

Converserly, if it does not coincide with Bayes discriminant error it assumes the form (see Problem 7.2):

$$P(f(\mathbf{x}, \mathcal{D})) = |2\gamma(\mathbf{x}) - 1| + P(y_b(\mathbf{x}) = y). \tag{7.7}$$

If we compute the mean over all data set of same cardinality  $\ell$ , we have:

$$P(f(\mathbf{x}, \mathcal{D}) \neq y) = |2\gamma(\mathbf{x}) - 1|P(f(\mathbf{x}, \mathcal{D}) \neq y_b) + P(y_b \neq y). \tag{7.8}$$

We call the term  $P(f(\mathbf{x}, \mathcal{D}) \neq y)$  *boundary error*, since it is the incorrect estimation of the optimal boundary [9]. The boundary error depends on  $P(f(\mathbf{x}, \mathcal{D}))$ , which is the probability of obtaining an estimate  $f(\mathbf{x})$  given a data set  $\mathcal{D}$ . If we assume that  $P(f(\mathbf{x}, \mathcal{D}))$  is a Gaussian, it can be shown [9] that the boundary error  $P(f(\mathbf{x}, \mathcal{D}) \neq y)$  is given by:

$$P(f(\mathbf{x}, \mathcal{D}) \neq y) = \Psi \left[ \text{sign} \left( \gamma(x) - \frac{1}{2} \right) \left( \mathcal{E}(f(\mathbf{x}, \mathcal{D})) - \frac{1}{2} \right) \sigma(f(\mathbf{x}; \mathcal{D}))^{-1} \right] \tag{7.9}$$

where *sign* is the signum function and  $\Psi(\cdot)$  is given by:

$$\Psi(u) = \frac{1}{2} \left[ 1 - \text{erf} \left( \frac{u}{\sqrt{2}} \right) \right]$$

and *erf*( $\cdot$ ) is the *error function*.<sup>1</sup>

In the Equation (7.9) we can identify two terms. The former, called *boundary bias term* ( $B_b$ ) [9], is represented by  $\text{sign} \left( \gamma(x) - \frac{1}{2} \right) \left( \mathcal{E}(f(\mathbf{x}, \mathcal{D})) - \frac{1}{2} \right)$ . The latter, called *variance term* ( $V_t$ ), is  $\sigma(f(\mathbf{x}; \mathcal{D}))^{-1}$ .

Therefore more concisely the equation can be (7.9) rewritten as:

$$P(f(\mathbf{x}, \mathcal{D}) \neq y) = \Psi [B_b V_b]. \tag{7.10}$$

In analogy with bias-variance decomposition in regression, we have represented the boundary error in classification in terms of boundary bias and variance. Whereas in regression the decomposition is simply additive, in the classification the decomposition is more complicated. The decomposition is nonlinear, due the presence of  $\Psi$  function, and multiplicative, since the argument of  $\Psi$  is given by the product of the boundary bias and the variance. Since the bias is expressed in terms of a signum function, it affects the boundary error in a limited way. Therefore the boundary error depends essentially on the

---

<sup>1</sup>  $\text{erf}(u) = \frac{2}{\sqrt{\pi}} \int_0^u e^{-u^2} du.$

variance. Conversely to the regression case, in classification it is fundamental to keep the variance as small as possible. On the contrary, the magnitude of boundary bias is not really important since only its signum is taken. This situation is expressed concisely in the sentence that in the classification *the variance dominates the bias*. In the next section we discuss another approach to characterize a learning machine that consists in measuring its *complexity*.

### 7.3 Model Complexity

In this section we introduce the concept of complexity in a learning machine or *model complexity*. In order to fix the ideas we consider a classification problem. In this case the data set (or *training set*) is formed by samples input-output, where to each input pattern is associated the desired output. A training set  $\mathcal{D}$  can be formalized as follows:

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$$

where the vectors  $\mathbf{x}_1, \mathbf{x}_\ell \in X \subset \mathbb{R}^n$  are called *patterns* and  $y_1, \dots, y_\ell$  take values in  $Y$ .  $Y = \{Y_1, \dots, Y_M\}$  is a discrete set, whose elements  $Y_i$  are called *classes*. The classification problem consists in finding a function  $f : X \rightarrow Y$ . We call this function *classifier*. The performance of the trained classifier is assessed measuring its capability to predict correctly a set of unseen data, called *test set*. Training and test sets are disjoint. The performances of the classifier on the training set is measured by the *training error* (or *empirical risk*) which is the average loss over the training set, that is:

$$Err_{train} = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathbb{L}(y_i, f(\mathbf{x}_i)) \quad (7.11)$$

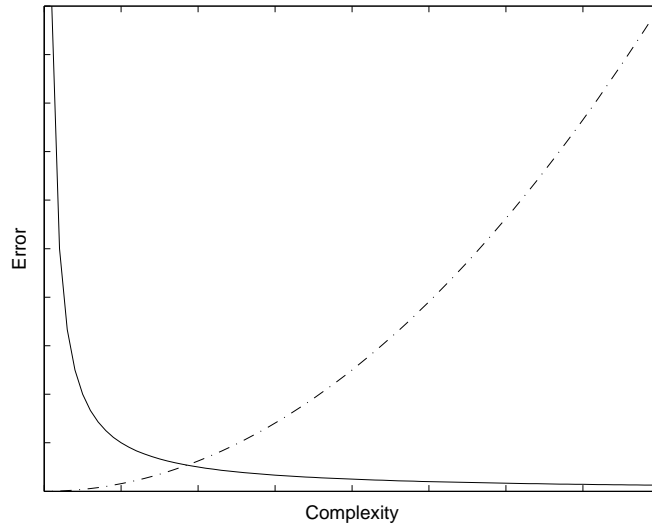
where  $y_i$  is the desired output (or *target*) for the pattern  $\mathbf{x}_i$  and  $f(\mathbf{x}_i)$  is the value computed by the classifier. A typical loss function is the *zero-one loss* (see Chapter 5). The loss is zero if the sample is classified correctly, one otherwise. We restrict our attention to the binary classification in which  $y$  can assume the conventional values  $\{1, -1\}$ . Hence the zero-one loss is:

$$\mathbb{L}(y_i, f(\mathbf{x}_i)) = \frac{1}{2} |y_i - f(\mathbf{x}_i)|. \quad (7.12)$$

and the training error becomes:

$$Err_{train} = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{1}{2} |y_i - f(\mathbf{x}_i)|. \quad (7.13)$$

Given a test set the classifier performances are measured by the *Test error* (or *generalization error* or *expected risk*), computed on test samples drawn on the basis of the underlying probability distribution  $P(\mathbf{x}, y)$ , that is:



**Fig. 7.1.** Qualitative behavior of  $E_{train}$  (solid curve) and  $E_{est}$  (dashed curve) in function of the complexity.

$$E_{test} = \mathcal{E}[\mathbb{L}(y, f(\mathbf{x}))] \quad (7.14)$$

where  $\mathbf{x}$  is a generic element of the test set and  $y$  the respective target. If we assume the zero-one loss function Equation (7.14) becomes:

$$E_{test} = \int \frac{1}{2} |y - f(\mathbf{x})| dP(\mathbf{x}, y) \quad (7.15)$$

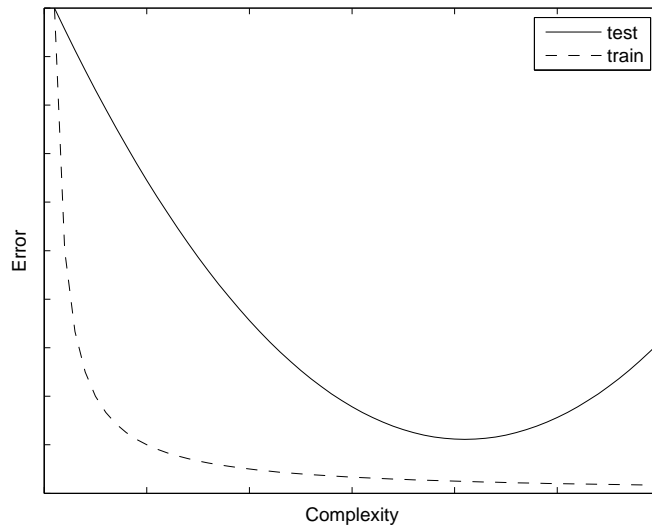
where we use the integral since the cardinality of test set can be infinite. In addition to the training and test error there is another quantity that characterizes the classifier, the so-called *complexity*.

Although the classifier complexity will be defined precisely in the next section, we assume which roughly depends on the number of parameters of the classifier. The higher is the number of the parameters the higher is its complexity. Being said that, we return to the training and test error and we observe that they are related by the following inequality:

$$E_{test} \leq E_{train} + E_{est} \quad (7.16)$$

where  $E_{est}$  is called *estimation error* (or *confidence term* or *capacity term*). Training and test error can differ significantly. Training error tends to decrease when the complexity of the classifier increases. On the other hand, the estimation error increases with the complexity increment, as shown in Figure 7.1. A classifier with no training error is usually not useful. Since it overfits the data,





**Fig. 7.2.** Qualitative behavior of the error on the training and test set in function of the classifier complexity.

it often performs poorly on the test set. The qualitative behavior of the error on the test set in function of the classifier complexity is shown in Figure 7.2. The curve above described represents qualitatively the generalization error in function of the complexity. In order to design accurate classifiers we need methods for estimating the test error curve quantitatively.

This chapter presents some methods for estimating the test error in function of the model complexity. The model usually has a vector of parameters  $\alpha$  that has to be set up in order to minimize the test error. We remark that we have two different goals. The first goal is estimating the performance of different models, i.e. with different values of  $\alpha$ , with the aim of picking the best one. This goal is called *model selection*. The second goal consists in estimating the generalization error, after having selected the final model. This goal is called *model assessment* [14].

If we have enough data, as it usually happens in handwriting recognition, a usual approach for model selection and assessment consists in dividing randomly data in three subsets: a *training set*, a *validation set* and a *test set*.

The training set is used to train the different models, i.e. the models with different values of  $\alpha$ . The validation set is used to estimate the generalization error for the models and to pick the best model. The test set is used to assess the test error of the selected model. The test set has to be used only for the model assessment. On the contrary, if we use the test set repeatedly, for instance in the phase of model selection, the model overfits the test set. In

this way, the test error of the selected model can underestimate notably the *real* generalization error. It is not possible to provide a general rule to assess the sizes of the training, validation and test set, since the size depends on the signal-to-noise ratio and the size of the overall data set. For instance, if the data set is very large a possible choice consists in dividing the data set in three equal parts.

In the rest of the chapter we will discuss the situation when the data are not enough to be divided in three sets. Even in this case there is no general criterion which permits deciding when data are adequate to be splitted in three sets. The adequate amount of data depends on the signal-to-noise ratio of the function that we want to approximate and the model complexity that we use for approximating the function. In this chapter we will describe methods that allow to choose the best model, without using the validation step. These models generally tries to estimate the optimal complexity. Finding the optimal complexity for a model is an example of the heuristics called *Occam's razor*,<sup>2</sup> proposed by the philosopher of the Middle Ages, William of Occam. According to the Occam's razor we should give the preference to simpler models instead of more complex ones. Therefore a model selection method should implement a trade-off strategy between the preference towards the simpler models and how much, expressed by the training error, we fit the data of the training set. This strategy is implemented by the model selection methods with the exception of crossvalidation, that we will describe in the chapter.

## 7.4 VC Dimension and Structural Risk Minimization

Statistical Learning Theory [3][22][23][24] provides a measure of the complexity of the classifier, the so-called *VC dimension* (or *Vapnik-Chervonenkis dimension* by the theory authors). In this section, following the approach of [14], we provide an intuitive definition of VC dimension, whereas a formal definition of VC dimension will be provided in the next section.

Consider a class of indicator functions  $\mathcal{C} = \{i(\mathbf{x}, \boldsymbol{\alpha})\}$  where  $i(\cdot)$  can assume only two values  $\{1, -1\}$  and  $\boldsymbol{\alpha}$  is a parameter vector. The VC dimension provides a method of measuring the complexity of the class of the function above defined. Before the definition of the VC dimension we introduce the following definitions.

**Definition 12** *A function separates perfectly a set of points if any point is classified correctly.*

**Definition 13** *A set of points is **shattered** by a class of functions  $\mathcal{C}$ , independently how the points are labeled, if an element of the class can perfectly separate them.*

Now we define the VC dimension.

<sup>2</sup> *Numquam ponenda sine necessitate* (W. Occam).

**Definition 14 (VC dimension)** *The VC dimension of the class of functions  $\mathcal{C}$  is defined as the largest number of points that can be **shattered** by elements of  $\mathcal{C}$ .*

The VC dimension is generally indicated by  $h$  and cannot exceed the number of samples of the training set  $\ell$ . Figure 7.3 shows that the VC dimension of the class of the linear function in  $\mathbb{R}^2$  is three. This result is generalized by the following theorem:

**Theorem 3 (Hyperplane VC Dimension)** *An hyperplane in  $n$  dimension has VC dimension equals to  $n + 1$ .*

We observe that for the hyperplane, its VC dimension coincides with the number of its free parameters. We remark that this does not generally happen for the other classes of functions. Now, we wonder if it exists a class of functions which has infinite VC dimension. The answer is provided by the following result [24]:

**Theorem 4** *The class of the functions  $\sin(\alpha x)$  has **infinite** VC dimension.*

The figure 7.4 shows an example in which a set of points can be shattered by the class of the function  $\sin(\alpha x)$  by choosing an appropriate value for  $\alpha$ . It is possible to prove that any set point can be shattered by the  $\sin(\alpha x)$  selecting a suitable  $\alpha$ .

After having defined the VC dimension, we quote the following result [22], for the binary classification, that put in connection the estimation error

**Theorem 5** *With probability  $1 - \eta$  (with  $\eta > 0$ ), the generalization error  $E_{test}$  is given by:*

$$E_{test} = E_{train} + E_{est} \quad (7.17)$$

where  $E_{train}$  is the error on the training set and  $E_{est}$  is given by:

$$E_{est} = \sqrt{\frac{1}{\ell} \left( h \left( \ln \frac{2\ell}{h} + 1 \right) + \ln \frac{4}{\eta} \right)} \quad (7.18)$$

An analogous result, for the regression, is reported in [5].

**Theorem 6** *With probability  $1 - \eta$  (with  $\eta > 0$ ), the generalization error  $E_{test}$  in the regression is given by<sup>3</sup>:*

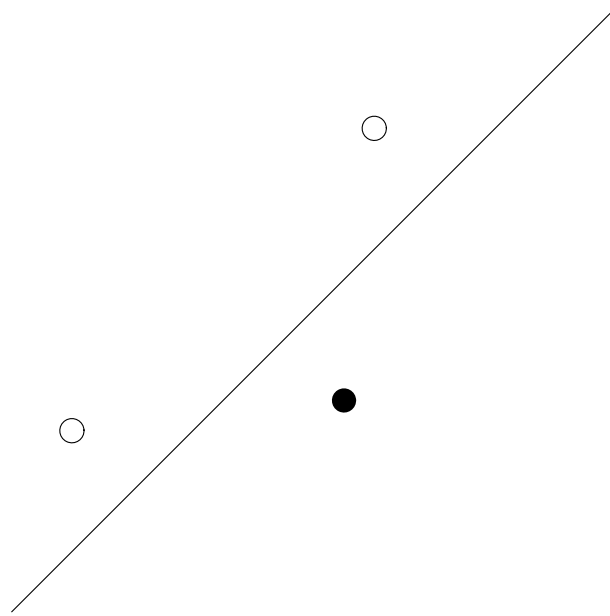
$$E_{test} = \frac{E_{train}}{(1 - c\sqrt{\epsilon})_+} \quad (7.19)$$

where  $E_{train}$  is the error on the training set and  $\epsilon$  is given by:

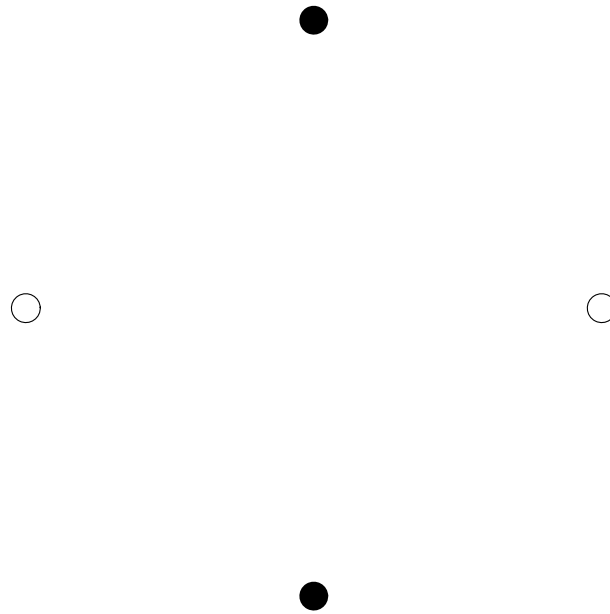
$$\eta = a_1 \frac{h(\log(a_2 \frac{\ell}{h}) + 1) - \log(\frac{\eta}{4})}{\ell} \quad (7.20)$$

with  $a_1, a_2, c \in \mathbb{R}$ .

<sup>3</sup>  $f(\cdot)_+$  stands for the positive part of  $f(\cdot)$

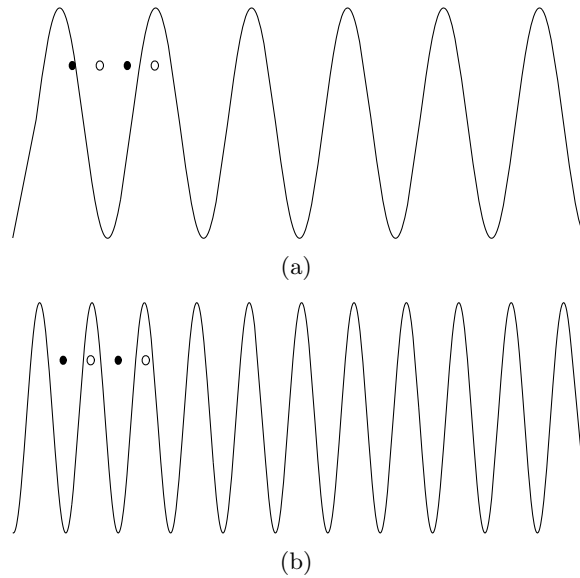


(a) A line can shatter three points.



(b) Four points cannot be shattered by a line.

**Fig. 7.3.** Three points can be shattered by the class of the lines in the plane, whereas four points cannot be shattered.



**Fig. 7.4.** (a) The set of points cannot be separated by  $\sin(\alpha x)$  using  $\alpha = 6$ ; (b) the same data set can be separated using choosing  $\alpha = 11$ .

Cherkassky and Mulier [5] suggest as typical values  $a_1 = a_2 = c = 1$ .

Now, we show how the VC dimension can be used for the model selection. The *structural risk minimization (SRM)*, proposed by Vapnik [26][24], is a model selection criterion based on the VC dimension. Structural Risk Minimization consists in training a sequence of models of increasing VC dimensions  $h_1 < h_2 < \dots < h_{p-1} < h_p < \dots$ . Then the model with smallest generalization error (provided by the Theorem 5) is picked. Unfortunately the bound on the generalization error provided by the theorem is very often too loose. In addition, it is not always possible to compute the VC dimension of a class of function. On the contrary, it can only compute an upper bound (often loose) for the VC-dimension. Therefore structural risk minimization generally results in a too imprecise criterion to be used as a model selection criterion.

## 7.5 Statistical Learning Theory\*

In this section we review some fundamental issues of *Statistical Learning theory*, also called *Vapnik-Chervonenkis theory* by the names of main contributors. The reading of this section can be omitted by readers not interested in the theoretical issues of learning.

Statistical Learning theory provides a mathematical framework for the learning problem. We assume that we have a data set

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \in X \times Y \quad (7.21)$$

whose samples are drawn according to an unknown underlying distribution function  $P(x, y)$ . The *learning problem* can be formalized in the following way.

**Definition 15 (Learning Problem)** *Learning consists in minimizing the expected loss, given by:*

$$\mathcal{R}[f] = \int_{X \times Y} \mathbb{L}(y, f(\mathbf{x})) dP(\mathbf{x}, y) \quad (7.22)$$

where  $\mathbb{L}(\cdot)$  is a loss-function (see Chapter 5). In the case of classification problem, a usual choice is to assume the *zero-one loss* as loss function.

The learning problem cannot be solved in a straight way. Since the probability density function is unknown, the integral in Equation (7.22) cannot be computed. Therefore it is necessary an alternative strategy to solve the learning problem. The strategy consists in replacing the expected risk with the *empirical risk*, computed on  $\mathcal{D}$ . Therefore we can define the following principle:

**Definition 16 Empirical Risk Minimization Principle (ERM)** *consists in choosing the function  $f(\cdot)$  that minimizes the empirical risk, given by:*

$$\mathcal{R}_{emp}[f] = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathbb{L}(y_i, f(\mathbf{x}_i)). \quad (7.23)$$

The ERM principle is theoretically sound, that is, *consistent*. The consistency of ERM principle means that  $\mathcal{R}_{emp}[f] \rightarrow \mathcal{R}[f]$  as the cardinality of the data set approaches the infinity, that is  $\ell \rightarrow \infty$ .

Now, we introduce a classical statistical inequality, the *Chernoff's bound* [6][8] that connects the empirical mean to the expected value of a variable.

**Theorem 7** *Let  $\xi_1, \dots, \xi_\ell$  be samples of a random variable  $\xi$ . For any  $\epsilon > 0$ , the following inequality, called **Chernoff's bound**, holds:*

$$P \left( \left| \frac{1}{\ell} \sum_{i=1}^{\ell} \xi_i - \mathcal{E}[\xi] \right| \geq \epsilon \right) \leq 2 \exp(-2\ell\epsilon^2). \quad (7.24)$$

Using Chernoff's bound [17], it can prove that the convergence of the empirical risk to the expected risk is exponential, that is the following result holds (see Problem 7.4):

**Theorem 8** *For any  $\epsilon > 0$ ,*

$$P(|\mathcal{R}_{emp}[f] - \mathcal{R}[f]| \geq \epsilon) \leq \exp(-2\ell\epsilon^2). \quad (7.25)$$

### 7.5.1 Vapnik-Chervonenkis Theory

Now we summarize the main issues of the Vapnik-Chervonenkis theory. We restrict our attention to the binary classification problem.

Let  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$  be a data set. Let  $\mathcal{F}$  be the class of the indicator functions, that is functions taking values in  $\{-1, 1\}$ , on  $\mathcal{D}$ . We denote with  $N(\mathcal{F}, \mathcal{D})$  the cardinality of  $\mathcal{F}$  restricted to  $\mathbf{x}_1, \dots, \mathbf{x}_\ell$ , namely the number of different separations of the data  $\mathbf{x}_1, \dots, \mathbf{x}_\ell$  by means of functions of the set  $\mathcal{F}$ . Besides, we denote with  $N(\mathcal{F}, \ell)$  the maximal number of separations can be produced in this way. The function  $N(\mathcal{F}, \ell)$  is called the *shattering coefficient*. Whenever the shattering coefficient is equal to  $2^\ell$ , all possible separations can be performed by  $\mathcal{F}$ . In this case we say that  $\mathcal{F}$  shatters  $\ell$  patterns. It is important to remark that  $\ell$  patterns means that it *exists a set of  $\ell$  patterns that can be separated*. It does not imply that each sets of  $\ell$  patterns can be separated.

Now we introduce three measures of capacity for the class  $\mathcal{F}$ , i.e. the *VC entropy*, the *annealed entropy* and the *growth function*. The *entropy* (or *VC entropy*) is defined as follows:

**Definition 17** *The VC entropy of the class function  $\mathcal{F}$  is defined by:*

$$H_{\mathcal{F}}(\ell) = \mathcal{E}[\ln N(\mathcal{F}, \mathcal{D})] \quad (7.26)$$

where the expectation  $\mathcal{E}[\cdot]$  is taken over  $\mathcal{D}$ .

The following result [24] connects the entropy to the consistency of the ERM principle:

**Theorem 9** *A sufficient condition for consistency of ERM principle is provided by*

$$\lim_{\ell \rightarrow \infty} \frac{H_{\mathcal{F}}(\ell)}{\ell} = 0. \quad (7.27)$$

The above result represents the *first milestone of VC theory* [24]. Any machine learning algorithm should satisfy (7.27).

The second measure of capacity is the *annealed entropy*.

**Definition 18** *The annealed entropy of the class function  $\mathcal{F}$  is defined by:*

$$H_{\mathcal{F}}^{ann}(\ell) = \ln \mathcal{E}[N(\mathcal{F}, \mathcal{D})]. \quad (7.28)$$

where the expectation  $\mathcal{E}[\cdot]$  is taken over  $\mathcal{D}$ .

The annealed entropy is an upper bound on the VC entropy [17] (see Problem 7.5). The following result (the former part is due to [24], the latter part is due to [4]) connects the annealed entropy to the rate of convergence of the empirical risk to the expected risk.

**Theorem 10** *If the annealed entropy [24] satisfies*

$$\lim_{\ell \rightarrow \infty} \frac{H_{\mathcal{F}^{ann}}(\ell)}{\ell} = 0 \quad (7.29)$$

*then for any  $\epsilon > 0$  the following equation holds:*

$$P \left( \sup_{f \in \mathcal{F}} |R[f] - R_{emp}[f]| > \epsilon \right) \leq 4 \exp \left( \frac{H_{\mathcal{F}^{ann}}(2\ell)}{\ell} - \epsilon^2 \right) \ell. \quad (7.30)$$

*Conversely [4], if condition (7.30) holds, then Equation (7.29) is fulfilled.*

Equation (7.29) represents the *second milestone of VC theory* [24] which guarantees a fast rate of convergence.

Now we can obtain an upper bound of the annealed entropy if we replace the expectation with the supremum over all possible samples. The new function is called *growth function*, that represents the third measure of capacity.

**Definition 19** *The growth function of the class function  $\mathcal{F}$  is defined by:*

$$G_{\mathcal{F}}(\ell) = \ln \sup_{\mathcal{D}} N(\mathcal{F}, \mathcal{D}). \quad (7.31)$$

We remark that the Vapnik-Chervonenkis' approach results in an upper bound on a set of classifiers and not a single classifier. Moreover, Vapnik and Chervonenkis use a *worst case approach*, due to the presence of supremum in (7.31). The following result [24] connects the growth function to the consistency of the ERM principle.

**Theorem 11** *A necessary and a sufficient condition for consistency of ERM principle is provided by*

$$\lim_{\ell \rightarrow \infty} \frac{G_{\mathcal{F}}(\ell)}{\ell} = 0. \quad (7.32)$$

*Besides, if the condition (7.32) holds, then the rate of convergence is given by (7.30).*

Equation (7.32) represents the *third milestone of VC theory* [24]. This milestone provides the necessary and sufficient condition that a learning algorithm implementing the ERM principle must fulfill in order to guarantee a fast rate of convergence independent of the problem that must be solved.

The following result [25] allows to define formally the VC dimension, that has been introduced informally in the previous section.

**Theorem 12 (VC Dimension's Theorem)** *The growth function  $G_{\mathcal{F}}(\ell)$  either satisfies the equality*

$$G_{\mathcal{F}}(\ell) = \ell \ln 2 \quad (7.33)$$

*or is given by:*

$$G_{\mathcal{F}}(\ell) \begin{cases} = \ell \ln 2 & \text{if } \ell \leq h \\ \leq h(1 + \frac{\ell}{h}) & \text{if } \ell > h \end{cases} \quad (7.34)$$



where  $h$ , called **Vapnik-Chervonenkis dimension (VC dimension)**, is the largest integer for which

$$G_{\mathcal{F}}(\ell) = h \ln 2 \quad (7.35)$$

If  $h$  does not exist, that is  $G_{\mathcal{F}}(\ell) = \ell \ln 2$ , VC dimension is said to be infinite.

## 7.6 AIC and BIC Criteria

In this section we describe two criteria for model selection, i.e. *Akaike information criterion (AIC)* [2] and *Bayesian information criterion (BIC)* [18]. These criteria are widely used when the number of the samples in the data set is small, typically less than 1000, as it often happens, for instance, in applications of time signal prediction or bioinformatics.

### 7.6.1 Akaike Information Criterion

The Akaike information criterion [2] can be used when the loss function of the model is a log-likelihood function, as happens in the models whose training is based on the maximum likelihood principle [9]. AIC consists of defining an index, called *AIC*, and in picking the model with smallest AIC. Let  $\{m_{\alpha}(\mathbf{x})\}$  be a class of models, where  $\alpha$  and  $\mathbf{x}$  are, respectively, the parameter vector that has to be tuned and  $\mathbf{x}$  is the input vector. If we denote with  $E_{train}(\alpha)$  and  $d(\alpha)$ , respectively, the error on the training set and the number of free parameters for each model, the AIC index, which is function of  $\alpha$ , is defined as follows:

$$AIC(\alpha) = E_{train}(\alpha) + 2 \frac{d(\alpha)}{\ell} \hat{\sigma}^2 \quad (7.36)$$

where  $\ell$  and  $\hat{\sigma}^2$  are, respectively, the number of samples of the training set and an estimate of the variance of the noise in the data.

A reasonable choice, provided by [10], for  $\hat{\sigma}^2$  is:

$$\hat{\sigma}^2 = \frac{E_{train}(\alpha)}{\ell - d(\alpha)}. \quad (7.37)$$

Plugging (7.37) in (7.36) we obtain the following expression, easy to compute, for AIC:

$$AIC(\alpha) = E_{train}(\alpha) + 2 \frac{d(\alpha) E_{train}(\alpha)}{\ell(\ell - d(\alpha))}. \quad (7.38)$$

The AIC index provides an estimate of the generalization error and we can use it for model selection. For this purpose, it is adequate to pick the model with the smallest AIC index.

Finally, we quote that a special case of the Akaike information criterion is the  $C_p$  statistics. More details can be found in [10][14].

### 7.6.2 Bayesian Information Criterion

*Bayesian Information Criterion (BIC)*, also called *Schwartz criterion* is similar to AIC. It can be used when the loss function of the model is a log-likelihood function. Likewise AIC, BIC defines an index, called *BIC* and picks the model with smallest BIC. If we use the same formalism defined in the section 7.6.1, the BIC index is defined as follows:

$$BIC(\boldsymbol{\alpha}) = E_{train}(\boldsymbol{\alpha}) + (\ln \ell) \frac{d(\boldsymbol{\alpha})}{\ell} \hat{\sigma}^2 \quad (7.39)$$

If we use for  $\hat{\sigma}^2$  the estimate given by (7.37), we obtain:

$$BIC(\boldsymbol{\alpha}) = E_{train}(\boldsymbol{\alpha}) + (\ln \ell) \frac{d(\boldsymbol{\alpha}) E_{train}(\boldsymbol{\alpha})}{\ell(\ell - d(\boldsymbol{\alpha}))}. \quad (7.40)$$

It is immediate to see that BIC is proportional to AIC. It is adequate to replace  $\ln \ell$  with 2 in (7.39) to get AIC. Since  $e^2$  is  $\sim 7.4$ , we have that it is reasonable that it is always  $\ln \ell > 2$ . This implies that BIC penalizes complex models more strongly than AIC. BIC chooses less complex models.

We conclude remarking that BIC can be motivated by a Bayesian approach to the problem of model selection. If we have a set of models  $\mathcal{S} = \{M_1, \dots, M_m\}$  and the respective model parameters  $\{\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_m\}$ . Our aim is to select the best model from  $\mathcal{S}$ . If we assume that we have a prior probability  $P(\boldsymbol{\alpha}_i|M_i)$  for the parameters of each model  $M_i$ , the posterior probability  $P(M_i|\mathcal{D})$ , by the Bayes Theorem, is:

$$P(M_i|\mathcal{D}) \propto P(M_i)P(\mathcal{D}|M_i) \quad (7.41)$$

where  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)$  is the training set.

It can be shown [14] that selecting the model with the smallest BIC index is equivalent to selecting the model with the largest posterior probability  $P(M_i|\mathcal{D})$ .

Besides, if we compute the BIC index for each model  $M_i$  and we denote with  $\beta_i$  the BIC index of the model  $M_i$ , it is possible to show that the posterior probability  $P(M_i|\mathcal{D})$  is given by:

$$P(M_i|\mathcal{D}) = \frac{\exp\left(\frac{\beta_i}{2}\right)}{\sum_{j=1}^m \exp\left(\frac{\beta_j}{2}\right)}. \quad (7.42)$$

Now, we compare BIC against AIC. Although it is not possible to assess in general which criterion is the best for the model selection, some considerations can be drawn. BIC is a *consistent model selection criterion*. This means that the probability that BIC picks the correct model tends to 1 as  $\ell \rightarrow \infty$ . On the contrary, AIC is *not consistent* since it selects models with too high complexity as  $\ell \rightarrow \infty$ . Finally, we remark that when the training set is finite BIC is often too parsimonious selecting model with too small complexity, due its large penalty term.

## 7.7 Minimum Description Length Approach

The *minimum description length (MDL)* [16] provides a model selection criterion based on the theory of coding.

From the viewpoint of the theory of coding, we can regard each pattern  $\mathbf{x}$  of data set as a message that we want to encode and to transmit to a *receiver*. We can view our model as a way of encoding the pattern. Therefore we will select the model that produces the shortest code.

Let  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\ell$  be the messages we want to send. The code uses a finite alphabet of length  $\Lambda$ . For instance, we can use a binary code. We can decide to encode our messages with a coding of variable length. In this case, if we use the strategy of *Huffman coding* (see Chapter 3) we will encode the most frequent messages with the shortest codes. Using Huffman coding the average message length is shorter.

In general it holds the following *Shannon's theorem*:

**Theorem 13** *If the messages  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\ell$  are transmitted respectively with probabilities  $P(\mathbf{x}_1), P(\mathbf{x}_2), \dots, P(\mathbf{x}_\ell)$ , the shortest coding uses code lengths  $\lambda_i = -\log_2 P(\mathbf{x}_i)$  and the average message  $E(\lambda)$  fulfills the following inequality:*

$$E(\lambda) \geq H. \quad (7.43)$$

Where  $H$ , called entropy of the distribution  $P(\mathbf{x}_i)$ , is given by:

$$H = -\sum_{i=1}^{\ell} P(\mathbf{x}_i) \log_2(P(\mathbf{x}_i)). \quad (7.44)$$

Besides, the equation (7.43) becomes an equality when the probabilities  $P(\mathbf{x}_i)$  are:

$$P(\mathbf{x}_i) = \Lambda^{-\lambda_i}$$

where  $\Lambda$  is the length of the alphabet.

We remark that when the set is infinite, the equation (7.44) has to be replaced with

$$H = -\int P(\mathbf{x}) \log_2(P(\mathbf{x})) d\mathbf{x}. \quad (7.45)$$

Therefore we can deduce the following corollary:

**Corollary 1** *In order to send a random variable  $\mathbf{x}$ , with probability density function  $P(\mathbf{x})$ ,  $-\log_2 P(\mathbf{x})$  bits of information are required.*

Finally, we can replace  $\log_2(P(\mathbf{x}))$  with  $\ln(P(\mathbf{x}))$ . This implies the introduction of the multiplicative factor  $\log_2 e$  that we can omit without mining the correctness of our arguments.

That being said, we can return to the model selection. Given a model  $\mathcal{M}$  having a parameter vector  $\boldsymbol{\alpha}$ , we denote with  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$  the training set. Let the conditional probability of the output be  $p(y|\boldsymbol{\alpha}, \mathcal{M}, \mathbf{x})$ .

Besides, we assume that all inputs are known by the receiver. The message length  $\lambda$  required to send the outputs to the receiver is:

$$\lambda = -\ln p(y|\boldsymbol{\alpha}, \mathcal{M}, \mathbf{x}) - \ln p(\boldsymbol{\alpha}|\mathcal{M}). \quad (7.46)$$

The first term of (7.46) represents the average code length for sending the difference between the model and the target values, whereas the second term represents the average code length for sending the model parameter vector  $\boldsymbol{\alpha}$ .

The MDL principle implies that the model that has to be selected is the one that minimizes (7.46). Equation (7.46) is the log-posterior distribution. Therefore, minimizing description length implies maximizing posterior probability. Since the BIC criterion is derived by the maximization of log-posterior probability, it is equivalent to MDL approach. BIC criterion can be considered as a tool for model selection based on MDL approach.

## 7.8 Crossvalidation

*Crossvalidation* [12][15][20] is one of the most popular model selection methods. The basic idea of crossvalidation, also called more properly *K-fold crossvalidation*, consists in using part of the training set to train the model and the remaining part of the training set to test the trained model. We pass to describe K-fold crossvalidation in detail. Let  $\ell$  be the number of samples of the training set. We divide the training set into  $K$  subsets with the same number of samples. Therefore each subset has approximately  $\frac{\ell}{K}$  samples. Then we train the model using data from  $K - 1$  subsets and test its performance on the remaining subsets. We repeat the process for each of  $K$  possible choices of the subset which is not used in the training. Then we compute the test error averaging over all  $K$  error.

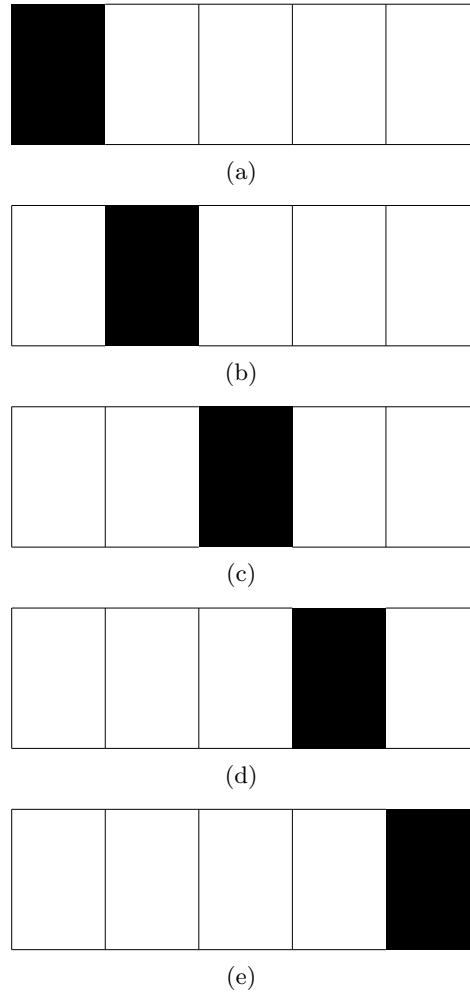
If we denote with  $Error_i(f(\mathbf{x}, \boldsymbol{\alpha}))$  the error on  $i^{th}$  subset of the model  $f(\mathbf{x}, \boldsymbol{\alpha})$ , the test error  $CV(\boldsymbol{\alpha})$  is given by:

$$CV(\boldsymbol{\alpha}) = \frac{1}{K} \sum_{i=1}^K Error_i(f(\mathbf{x}, \boldsymbol{\alpha})). \quad (7.47)$$

The crossvalidation picks the model with the parameter  $\boldsymbol{\alpha}$  which minimizes  $CV(\boldsymbol{\alpha})$ . Finally, the selected model is trained again on the whole data set. Typical values for  $K$  is 5 or 10 [14]. The case  $K = \ell$  is called *leave-one-out crossvalidation* [21]. In this case the model is trained using all patterns with the exception of one pattern.

### 7.8.1 Generalized Crossvalidation

For linear models that use the minimum square error as a loss function, leave-one-out crossvalidation can be approximated by *Generalized crossvalidation* (or



**Fig. 7.5.** Schematic representation of 5-fold crossvalidation. The data are divided into five segments. The model is trained five times, each time using a data set in which one of the subset (shown in black) is left out.

*GCV* [7]. Let  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$  be a dataset, where  $\mathbf{x} \in \mathbb{R}^n$  and the generic element  $y_i$  is the target value for  $\mathbf{x}_i$ . Let  $Y = (y_1, \dots, y_\ell)$  be the vector whose components are the target values  $y_i$ . Besides, we indicate with  $f(\mathbf{x}_i)$  the output of a linear model  $\mathcal{M}$  having as input the pattern  $\mathbf{x}_i$  and with  $F = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_\ell))$ . If  $\mathcal{M}$  is linear, it is possible to write the following equation:

$$Y = SF \quad (7.48)$$

where  $S$  is an  $\ell \times \ell$  matrix which depends on the input pattern  $\mathbf{x}_i$  but not on the targets  $y_i$ .

The GCV index is defined as follows:

$$GCV = \frac{1}{\ell} \sum_{j=1}^{\ell} \left[ \frac{y_j - f(\mathbf{x}_j)}{1 - \frac{\text{trace}(S)}{\ell}} \right]^2 \quad (7.49)$$

where  $\text{trace}(S)$  (with  $\text{trace}(S) < \ell$ ) is the sum of the diagonal elements of  $S$  and is called the *effective number of parameters*.

GCV can be preferred to leave-one-out crossvalidation when the  $\text{trace}(S)$  can be computed easily. Finally, we conclude pointing out that other model selection methods are based on effective number of parameters [24]. Among them, we quote *finite prediction error* [1] and *Shibata's model selector* [19].

## 7.9 Conclusion

In this chapter we have provided an overview of the main issues of statistical learning and model selection theories. We have discussed the problem of how to select the best one among a set of learning machines. Firstly, we have discussed the bias-variance showing how it can describe the behavior of a learning machine on the basis of simple statistical considerations. Then we have introduced the concept of the complexity of a learning machine presenting both intuitively and formally the most popular measure of complexity of a classifier that is the VC dimension. We have introduced the ERM principle and reviewed the main results of the Vapnik-Chervonenkis theory of learning, underlining the conditions that a learning machine has to fulfill in order to guarantee the consistency and the fast convergence of the ERM principle. The rest of the chapter has been devoted to review the most popular model selection methods that is BIC, AIC and crossvalidation. We have also briefly reviewed the minimal description length approach to the model selection underlining its equivalence to the BIC criterion.

We conclude the chapter providing some bibliographical remarks. Bias-variance decomposition is fully discussed in [9]. A comprehensive survey of the Vapnik-Chervonenkis theory can be found in [17][22][23][24]. Model Selection methods are described in detail in [14].

## Problems

**7.1.** Prove that the average error, in the case of regression,  $\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x}))^2]$  can be decomposed in the following way:

$$\mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x}))^2] = (\mathcal{E}[f(\mathbf{x}, \mathcal{D}) - F(\mathbf{x})])^2 + \mathcal{E}[(f(\mathbf{x}, \mathcal{D}) - \mathcal{E}[f(\mathbf{x}, \mathcal{D})])^2]$$

**7.2.** Consider the bias-variance decomposition for classification. Show that if the classification error  $P(f(\mathbf{x}, \mathcal{D}) = y)$  does not coincide with Bayes discriminant error, it is given by:

$$P(f(\mathbf{x}, \mathcal{D}) = y) = |2\gamma(\mathbf{x}) - 1| + P(y_b(\mathbf{x}) = y).$$

**7.3.** Prove that the class of functions  $\sin(\alpha x)$  ( $\alpha \in \mathbb{R}$ ) has infinite VC dimension (Theorem 4). You can compare your proof with the one reported in [24].

**7.4.** For any  $\epsilon > 0$ , prove that

$$P(|\mathcal{R}_{emp}[f] - \mathcal{R}[f]| \geq \epsilon) \leq \exp(-2\ell\epsilon^2) \quad (7.50)$$

**7.5.** Prove that the annealed entropy is an upper bound of VC Entropy. *Hint:* use Jensen's inequality [24] which states that for a concave function  $\psi$  the inequality

$$\int \psi(\Phi(x))dF(x) \leq \psi\left(\int \Phi(x)dF(x)\right)$$

holds.

**7.6.** Prove that if a class of function  $\mathcal{F}$  can shatter any data set of  $\ell$  samples the third milestone of VC theory is not fulfilled, that is the condition (7.32) does not hold.

**7.7.** Implement the AIC criterion. Consider *spam data* that can be downloaded by <ftp.ics.uci.edu/pub/machine-learning-databases/spam>. Divide randomly spam data in two subsets with the same number of samples. Take the former and the latter sets respectively as the training and the test set. Select a learning algorithm for classification (e.g. K-Means or MLP) and train the algorithm with several parameter values. Use the AIC criterion for model selection. Compare their performances by means of the model assessment.

**7.8.** Implement the BIC criterion. Repeat Problem 7.7 and use the crossvalidation for model selection. Compare its performance with AIC.

**7.9.** Implement the crossvalidation criterion. Repeat Problem 7.7 and use 5-fold crossvalidation for model selection. Compare its performance with AIC and BIC.

**7.10.** Implement the leave-one-out method and test it on *Iris Data* [11] which can be downloaded by <ftp.ics.uci.edu/pub/machine-learning-databases/iris>.





---

## References

1. H. Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 21:202–217, 1970.
2. H. Akaike. Information theory and an extension of the maximum likelihood principle. In *2<sup>nd</sup> International Symposium on Information Theory*, pages 267–281, 1973.
3. M. Anthony. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
4. S. Boucheron, G. Lugosi, and S. Massart. A sharp concentration inequality with applications. *Random Structures and Algorithms*, 16(3):277–292, 2000.
5. V. Cherkassky and F. Mulier. *Learning from Data*. John Wiley, 1998.
6. H. Chernoff. A measure of asymptotic efficiency of tests of a hypothesis based on the sum of observations. *Annals of Mathematical Sciences*, 23:493–507, 1952.
7. P. Craven and G. Wahba. Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized crossvalidation. *Numerische Mathematik*, 31(4):377–403, 1978.
8. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.
9. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
10. B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.
11. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
12. K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.
13. S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias-variance dilemma. *Neural Networks*, 4(1):1–58, 1992.
14. T. Hastie, R.J. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.
15. F. Mosteller and J.W. Tukey. Data analysis, including statistics. In *Handbook of Social Psychology*, pages 80–203. Addison-Wesley, 1968.
16. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431, 1983.
17. B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.

18. G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.
19. R. Shibata. An optimal selection of regression variables. *Biometrika*, 68(1):45–54, 1981.
20. M. Stone. Cross-validated choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, B36:111–147, 1974.
21. M. Stone. An asymptotic equivalence of choice of model by crossvalidation and akaike’s criterion. *Journal of the Royal Statistical Society*, B39:44–47, 1977.
22. V.N. Vapnik. *Estimation of Dependences based on Empirical Data*. Springer-Verlag, 1982.
23. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.
24. V.N. Vapnik. *Statistical Learning Theory*. John Wiley, 1998.
25. V.N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.
26. V.N. Vapnik and A. Ya. Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974.

## Supervised Neural Networks and Ensemble Methods

### What the reader should know to understand this chapter

- Fundamentals of machine learning (Chapter 4).
- Statistics (Appendix A).

### What the reader should know after reading in this chapter

- Multilayer neural networks.
- Learning vector quantization.
- Classification and regression methods.
- Ensemble methods.

## 8.1 Introduction

In supervised learning, the data is a set  $\mathcal{D}$  whose elements are input-output patterns, i.e.

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \in \mathbb{R}^d \times \mathcal{Y} \quad (8.1)$$

and the learning problem can be thought as finding a function  $f : \mathbb{R}^d \rightarrow \mathcal{Y}$  that maps the vectors  $\mathbf{x}$  into the elements of  $\mathcal{Y}$ . If the set  $\mathcal{Y}$  is discrete, i.e.  $\mathcal{Y} = \{\mathcal{C}_1, \dots, \mathcal{C}_K\}$  the learning problem is called *classification*. An example of this learning task is the recognition of handwritten digits or a speaker. Such a task is performed with algorithms called *classifiers* (see Chapter 5).

If the set  $\mathcal{Y}$  is continuous, i.e.  $\mathcal{Y} \subseteq \mathbb{R}^K$ , the problem is called *regression*. Example of this learning task is the prediction of stock indexes. Such a task is performed with algorithms called *regressors*.

This chapter presents some learning algorithms that have the peculiarity of being *supervised* (see Chapter 5), i.e. of being capable to learn from a set of input-output examples  $\mathcal{D}$  called *training set*. In particular, this chapter focuses on three kinds of algorithms: artificial neural networks, learning vector quantization, and the ensemble methods.

The artificial neural networks implement a computational paradigm inspired by the anatomy of the brain. The corresponding algorithms simulate simple processing units (the so-called *neurons*) linked through a complex web of *connections*. This enables the networks to process separately different pieces of information while keeping into account their mutual constraints and relationships. The learning vector quantization is a supervised *prototype-based classifier*. Several clustering methods presented in Chapter 6, e.g. K-Means and SOM, can be viewed as prototype-based classifiers, when they are used in the classification task. However thanks to the information of the membership (or non-membership) of a pattern to a given class, LVQ outperforms unsupervised prototype-based classifiers. The ensemble methods are techniques that combine the output of a set of individually trained learning algorithms  $f_i(\mathbf{x})$  in order to obtain a performance higher than the performance of any single  $f_i(\mathbf{x})$ .

The rest of this chapter is organized as follows: Section 8.2 presents the general aspects of artificial neural networks, Sections 8.3 and 8.4 present artificial neurons and connections respectively, Section 8.5 shows single layer neural networks, while Sections 8.6 and 8.7 present multiple layer networks and their training algorithms respectively. In the last part of the chapter, Section 8.8 describes the learning vector quantization and Section 8.9 presents the Ensemble methods; finally some bibliographical remarks are provided in Section 8.10.

## 8.2 Artificial Neural Networks and Neural Computation

Consider an everyday action as simple as grabbing an object on a desk. Its execution involves the simultaneous processing of many pieces of information: the position of the object on the desk, the presence of obstacles, the identification of the object in the visual field, an approximate prediction of object weight and distance, etc. Each information piece can be partial or ambiguous, but still it can have a non negligible impact on the outcome of the overall process. Moreover, the single pieces of information cannot be processed separately, but must be considered as elements of a complex web of relationships. This means that the meaning and the role of the same information piece can change significantly depending on the connections with other information at hand [28]. The solution adopted by the nature for such a problem can be observed in the structure of the brain. In very simple terms (for a more rigorous description see [34]), the brain is composed of a large number of *neurons*,  $\sim 10^{11}$  in the case of humans, connected with each other through an even larger number of *synapses*, which is  $\sim 10^{14}$  in the case of humans. These carry signals, mainly in the form of electric or chemical stimulations, that are distributed to different neurons and separately elaborated by each one of them. The result of such a process is a collective behavior pattern enabling

the brain to perform all kinds of complex tasks, including the reading of this text.

The above description is the basis of a paradigm referred to as *neural computation* [18][20], *Parallel Distributed Processing* [42], *neurocomputing* [19] or *connectionism* [31] which aims at carrying out computational tasks by using a large number of simple interconnected processing units called *neurons* or *nodes*. These can be implemented through software simulations or hardware circuits and perform relatively simple calculations. The resulting machines are called *artificial neural networks* (ANN) and have an important characteristic: the connections between neurons are associated with parameters called *weights* that can be modified, through a training process, in order to associate a desired output to a given input. In other words, the ANNs can learn from input-output examples how to associate the correct output to previously unseen input data, and this is useful in the context of classification and regression problems.

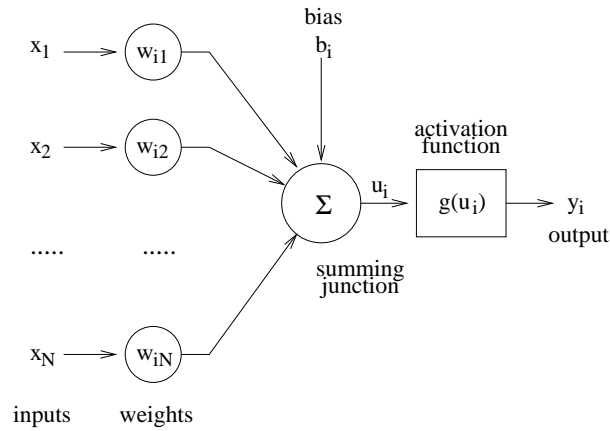
The neural networks have some important advantages with respect to other approaches [18][27][28]:

- *Nonlinearity.* When the neurons process the data with nonlinear functions, the networks as a whole are nonlinear. This is especially suitable when the mechanisms generating the data are inherently nonlinear.
- *Input output mapping.* The networks learn by adapting their parameters in order to map labeled input vectors  $\mathbf{x}_i$  to desired outputs  $\mathbf{t}_i$ , which are often called *targets*. This means that no assumption is made about the distribution of the data and the networks can perform *non-parametric statistical inference*.
- *Adaptivity.* The training process does not depend on the data. The learning properties are inherent to the networks and the same network can be trained to perform different tasks by simply using different data in the training. Nothing must be changed in the network to do so.
- *Contextual information.* Each neuron is affected by any other neuron, then contextual information is naturally used in the computation.

The next sections show in more detail the elements outlined above. In particular, after a description of neurons and connections, the chapter shows that the linear discriminant functions (see Chapter 5) can be thought of as neural networks and presents the most important example of ANN, i.e. the multilayer perceptron.

## 8.3 Artificial Neurons

The most general form of artificial neuron is depicted in Figure 8.1. Each neuron  $i$  in a network receives several inputs passing through connections characterized by weights  $w_{ik}$  (represented as circles in the figure). Each input value is multiplied by the weight of the connection it passes through and it



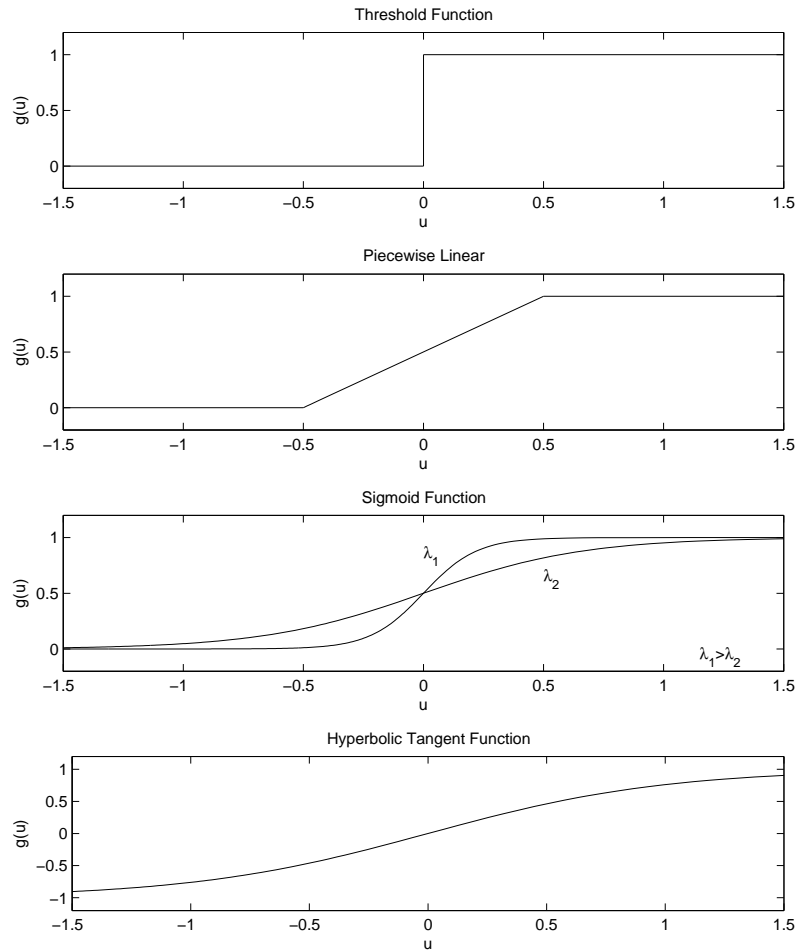
**Fig. 8.1.** Artificial neurons. This figure shows the most general form of artificial neurons. The inputs, multiplied by the connection weights, pass through a summing junction and the result is given as input to an activation function that gives the neuron output.

is conveyed to a junction (denoted with  $\Sigma$  in the figure) where all inputs are summed. A further term, called *bias* is added to the sum and the result is:

$$u_i = \sum_{k=1}^N w_{ik}x_k + b_i = \mathbf{w}_i \cdot \mathbf{x} + b_i \quad (8.2)$$

where  $\mathbf{w}_i$  is the vector having as components the weights of the connections ending in neuron  $i$  and  $\mathbf{x}$  is the vector of the inputs of the same neuron. Note that the input is higher than zero when  $\mathbf{w}\mathbf{x} > -b_i$  and this explains the role of the bias. In fact, the functions which determine the output of the neurons (see below) mimic a more or less abrupt transition from quiet to activity in correspondence of  $u_i = 0$ . The opposite of the bias can then be thought of as a threshold to be reached for activation.

The value  $u_i$  is given as input to an *activation function*  $g(u_i)$  which provides the output  $y_i$  of the neuron. The name activation function comes from an analogy with real neurons. In the brain, neurons behave roughly as electric condensers: they accumulate potential by receiving electric charges from their synapses and then discharge when the potential exceeds a threshold. The activation functions (see below for more details) mimic such a behaviour using both linear and nonlinear and nonlinear functions that are zero or close to zero up to a certain  $u_i$  value (conventionally fixed at  $u_i = 0$ ) and then grow more or less quickly to 1. If all activation functions in a neural network are linear, the network as a whole is a linear function. On the other hand, even if only part of the network neurons have a nonlinear activation function, the network as a whole is nonlinear.



**Fig. 8.2.** Activation functions. The plots show different activation functions commonly applied in neural networks. From top to bottom the functions are step, piecewise linear, sigmoid, and hyperbolic tangent.

The most common activation functions are the *step function* (or *Heaviside function* or *threshold function*), the *piecewise linear function*, the *logistic sigmoid* and the *hyperbolic tangent* (see Figure 8.2). All functions have the same basic behavior, but they have different properties that have an impact not only on the final results, but also on the training algorithms (see Section 8.7.3). The single functions are described more in detail in the following.

The step function  $I(u)$  is defined as follows:

$$I(u) = \begin{cases} 0 & \text{for } u < 0 \\ 1 & \text{for } u \geq 0 \end{cases} \quad (8.3)$$

and it is shown in the upper plot of Figure 8.2. Such an activation function was proposed in the earliest works on neuron models [29] and processing nodes such that  $g(u) = I(u)$  are sometimes referred to as *McCulloch-Pitts* neurons from the name of the model proposers.

A smoother version of the step function is the piecewise linear function defined as follows:

$$L(u) = \begin{cases} 0 & \text{for } u < \frac{1}{2} \\ u + \frac{1}{2} & \text{for } \frac{1}{2} \leq u \leq \frac{1}{2} \\ 1 & \text{for } u > \frac{1}{2}. \end{cases} \quad (8.4)$$

(see second plot from above in Figure 8.2). In this case, the transition is less abrupt and enables a gradual transition towards the activation.

The first two functions are simple, but are not continuous and this creates some problems for the training algorithms, then other functions have been proposed that have a similar shape, but are continuous. The first one is the *logistic sigmoid*:

$$\sigma(u) = \frac{1}{1 + e^{-\lambda u}} \quad (8.5)$$

where  $\lambda$  is called *slope parameter*. The higher  $\lambda$ , the steeper the transition from zero to one (see third plot from above in Figure 8.2). One of the main advantages of the sigmoid function is that it can be interpreted as a probability and this is often helpful in interpreting the output of a neural network (see Section 8.5.2).

The last function presented here is the *hyperbolic tangent*:

$$\Sigma(u) = \tanh(u) = \frac{e^{\lambda u} - e^{-\lambda u}}{e^{\lambda u} + e^{-\lambda u}} \quad (8.6)$$

which is shown in the lowest plot of Figure 8.2. An important difference with respect to the other functions is that the hyperbolic tangent takes values in the interval  $[-1, 1]$  rather than in the interval  $[0, 1]$ . The functions  $\sigma(u)$  and  $\Sigma(u)$  are related through a linear transform:

$$\Sigma(\tilde{u}) = 2\sigma(u) - 1 \quad (8.7)$$

where  $\tilde{u} = u/2$ . A neural networks having logistic sigmoids as activation functions is equivalent to a neural network having hyperbolic tangents as activation functions, but different values for weights and biases. The networks using the hyperbolic tangent are empirically found to converge faster than those using the logistic sigmoid [1].

The neurons are the first important element of a network, but they are not effective if they are not connected with each other. The connections play not only the role of channels through which the information flows, but they define also the architecture of the network. The next section shows in more detail how this happens.



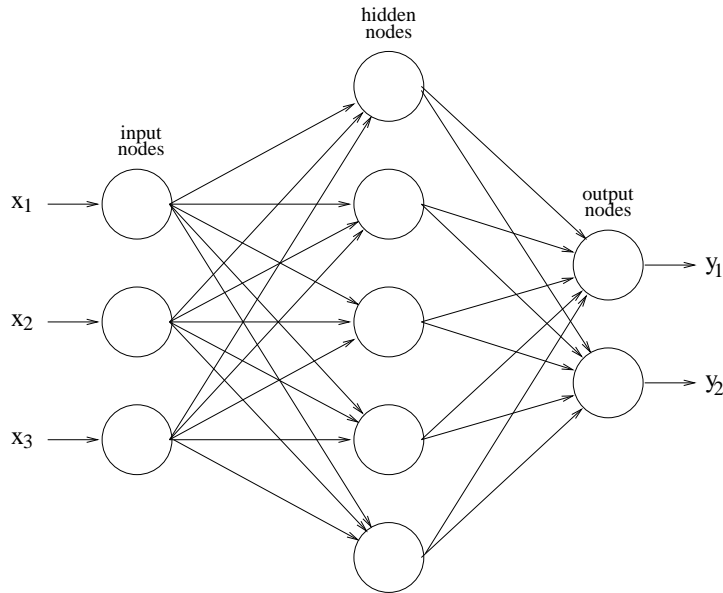
## 8.4 Connections and Network Architectures

Section 8.2 shows that the neural computation paradigm addresses the problem of processing a large amount of information pieces related to each other through contextual constraints. The neurons are the solution proposed for the first part of the problem, i.e. the handling of multiple and localized information elements. In fact, it is possible to feed each neuron with a single piece of information and to have a number sufficiently large of neurons to process the whole information at hand. On the other hand, since neurons focus on single and localized pieces of information, they cannot account for the relationships with the other information pieces and such a problem is rather addressed by the other important element of the neural networks, i.e. the connections.

The connections include two main aspects: the first is the architecture of the network, i.e. the fact that by connecting certain neurons rather than others the networks assume different structures. The second is the value of the weights associated to each connection. In principle, each neuron can be connected to any other neuron, but this book will focus on the so-called *feed-forward* networks, i.e. to networks where there are no feed-back loops. This means that the neurons can be grouped into disjoint sets  $S_i$ , where  $i \in (1, \dots, S)$ , such that all neurons belonging to set  $S_i$  receive inputs only from the neurons of set  $S_{i-1}$  and send their output only to the neurons of set  $S_{i+1}$ .

Figure 8.3 shows the *multilayer perceptron*, probably the most important example of feed-forward neural network. The figure clearly shows that there are three sets of neurons with the above outlined property. The neurons of the first set are called *input nodes* and, in general, they do not perform any kind of processing, i.e. their outputs simply correspond to a component of the input vector  $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{R}^d$ . On the contrary, the neurons of the other two sets, called *hidden* and *output* nodes, process their input as described in Section 8.3. The sets of neurons identified following the above approach are often called *layers*. The name hidden denotes the layers which are neither input nor output. A network can have more than one hidden layers. The network of the figure has three layers since it has only one hidden layer. However, other naming conventions propose to consider the connections rather than the neurons as elements of the layers, then the network of the figure would have only two layers. The reason behind such a choice is that what actually characterizes the network are the connections and not the nodes (see below for more details) and this book will adopt for this reason the second convention. When all neurons of set  $S_i$  are connected to all neurons of set  $S_{i+1}$ , the network is said *fully connected*.

The second important aspect of the connections is the value of the weights associated to them. The connection between neurons  $i$  and  $k$  is typically denoted with  $w_{ki}$ , meaning that the connection carries the output of neuron  $i$  into neuron  $k$  and the whole set of weights and biases (see Section 8.3) is typically denoted with  $\mathbf{w}$  and called *parameters set*. The value of weights



**Fig. 8.3.** Multilayer perceptron. The picture shows a fully connected multilayer perceptron.

and biases is determined through a supervised learning process aimed at finding the parameters set  $\tilde{\mathbf{w}}$  satisfying some predefined criterion. The value of weights and biases can then be thought of as the form under which is stored the knowledge acquired during the training [18][40].

Such an aspect is particularly important because a network with given architecture and activation functions can be trained to perform different tasks. In fact, it is sufficient to train the network with different data and the weights will assume the values that better correspond to each task. The connections determine the relationships between the different pieces of information processed by single neurons. Negative weights determine inhibitory effects of one piece of information onto another one, while positive weights correspond to excitatory effects.

So far, we have described the neural networks in intuitive terms using the similarity with the brain and giving a high level sketch of the way they work. The next sections show how the intuitive concepts outlined so far are translated into mathematical terms and how neural networks can be used to take decisions about the data and solve supervised problems.

## 8.5 Single-Layer Networks

This section shows how *linear discriminant functions* (LDF) [33] (see Chapter 5), a simple approach for the classification problem, can be interpreted as

single layer networks, i.e. neural networks with a single layer of connections (see Section 8.4 for the naming convention). Attention will be mainly paid to the way networks can perform classification tasks, for a rigorous and complete description of LDFs the reader can refer to most of the machine learning books (see e.g. [33]).

The rest of this section shows in particular that the neuron model presented above corresponds to a binary LDF (Section 8.5.1), that the logistic sigmoid function estimates a-posteriori class probabilities (Section 8.5.2), and that single layer networks can account only for linear separation surfaces between classes (Section 8.5.3).

### 8.5.1 Linear Discriminant Functions and Single-Layer Networks

Consider the problem of the binary classification, i.e. of the assignment of an input vector  $\mathbf{x}$  to one of two predefined classes  $\mathcal{C}_1$  and  $\mathcal{C}_2$ . Among other techniques (see Chapter 5 for Bayesian approaches), it is possible to use a discriminant function  $y(\mathbf{x})$  with the following property:

$$\begin{aligned} y(\mathbf{x}) &> 0 \text{ if } \mathbf{x} \in \mathcal{C}_1 \\ y(\mathbf{x}) &< 0 \text{ if } \mathbf{x} \in \mathcal{C}_2. \end{aligned} \quad (8.8)$$

The LDF is the simplest function of such kind and, in its most general form, is written as follows:

$$y(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x} + w_0), \quad (8.9)$$

where  $\mathbf{w}$  is a parameters vector of the same dimension  $d$  as  $\mathbf{x}$ ,  $w_0$  is a parameter called *bias* or *threshold*, and  $g(\cdot)$ , in the most simple case, is the identity function:

$$y(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0. \quad (8.10)$$

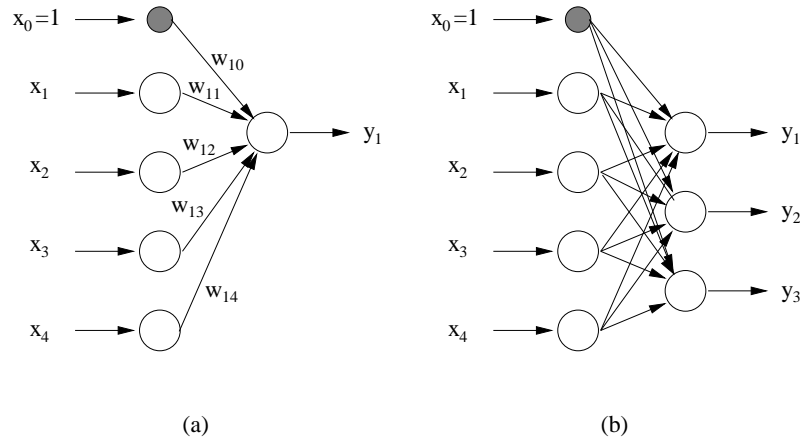
The set of the points where  $y(\mathbf{x}) = 0$  is called *separation surface* because it separates the regions corresponding to the two classes. If two points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  belong to the separation surface, then  $\mathbf{w} \cdot \mathbf{x}_1 + w_0 = \mathbf{w} \cdot \mathbf{x}_2 + w_0$  and:

$$\mathbf{w}(\mathbf{x}_1 - \mathbf{x}_2) = 0, \quad (8.11)$$

i.e. the parameters vector  $\mathbf{w}$  is orthogonal to the separation surface. Since  $\mathbf{w}$  is constant, the separation surface must be a hyperplane, hence the name Linear Discriminant Function. Equation (8.10) corresponds to the network in Figure 8.4 (a) when  $g(\cdot)$  is the identity, in fact it can be rewritten as:

$$y(\mathbf{x}) = \sum_{i=1}^d w_i x_i + w_0, \quad (8.12)$$

i.e. the input of a neuron as proposed in Equation (8.2) if we interpret  $w_0$  as the bias.



**Fig. 8.4.** Linear discriminant functions. The left network corresponds to a binary classifier of the kind described in Equation 8.9. The dark neuron corresponds to an extra input ( $x_0 = 1$ ) which enables one to account for the threshold  $w_{10}$ . The right network corresponds to the multiclass case.

Consider now the case where the number of classes is  $K$ . The problem can be addressed by using  $K$  binary classifiers  $y_i(\mathbf{x})$  capable of discriminating between vectors belonging to  $\mathcal{C}_i$  and vectors not belonging to  $\mathcal{C}_i$ :

$$\begin{aligned} y_i(\mathbf{x}) &> 0 \text{ if } \mathbf{x} \in \mathcal{C}_i \\ y_i(\mathbf{x}) &< 0 \text{ if } \mathbf{x} \notin \mathcal{C}_i. \end{aligned} \quad (8.13)$$

The class of an input vector  $\mathbf{x}$  can then be identified as follows:

$$k = \arg \max_i y_i(\mathbf{x}) = \arg \max_i \sum_{l=1}^d w_{il} x_l + w_{i0}. \quad (8.14)$$

This corresponds to the network depicted in Figure 8.4 (b) when the weights  $w_{l0}$ ,  $l \in (1, \dots, K)$ , are set to one. The single layer networks are then capable of performing classification tasks, although they are affected by the same limitations as the LDFs, i.e. they can account only for linear separation surfaces. The problem of training such a network is addressed in Section 8.7. Note that this technique does not make any assumption about the distribution of the data, then it belongs to the family of non-parametric methods.

### 8.5.2 Linear Discriminants and the Logistic Sigmoid

This section considers the case where the probabilities  $p(\mathbf{x}|\mathcal{C}_k)$  are Gaussians:

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right] \quad (8.15)$$

and the covariance matrices of different classes are equal. In the case of the binary classification, by the Bayes theorem:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \frac{1}{1 + \exp(-u)} = g(u) \quad (8.16)$$

where

$$u = \ln \left[ \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \right] \quad (8.17)$$

and  $g(u)$  is nothing else than the logistic sigmoid introduced in Section 8.3. If we pose  $u = \mathbf{w}\mathbf{x} + w_0$ , then  $g(u)$  corresponds to Equation (8.9) and:

$$\mathbf{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \quad (8.18)$$

$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2 + \ln \frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}. \quad (8.19)$$

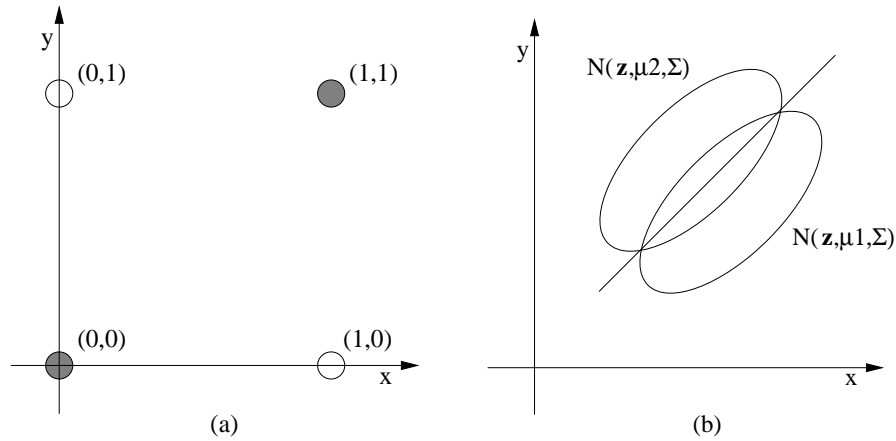
This corresponds to a network like the one depicted in Figure 8.4 (a) where the activation function is a logistic sigmoid. The multiclass case can be obtained by simply considering, like in the previous section, several binary classifiers.

The above has two main consequences: the first is that the parameters  $\mathbf{w}$  and  $w_0$  can be estimated with averages and covariances of the training data, then we have a technique to train a linear discriminant classifier and the corresponding neural network. The second is that the output of the nodes where the activation function is a logistic sigmoid can be thought of as a-posteriori probabilities of different classes. This is important because it enables one to interpret the networks output and to include it in probabilistic frameworks.

### 8.5.3 Generalized Linear Discriminants and the Perceptron

The main limit of the linear discriminant functions of Equation (8.9), and of the corresponding networks, is that they account for a narrow class of possible discriminant functions which, in many cases, are not the optimal choice. In fact, Section 8.5.1 shows that the separation surfaces implicitly identified by single layer networks are hyperplanes, then the LDFs are effective only in problems where different classes can be actually separated by linear surfaces. An example often presented in the literature [32] where single layer networks fail in separating two classes is the so-called *XOR problem* shown in Figure 8.5 (a). In this case, no linear surface can separate the samples belonging to the two classes. On the other hand, the linear separation surface is optimal (in the sense of the error rate minimization) in the case of two partially overlapping classes following Gaussian distributions as shown in Figure 8.5 (b). Since they are simple and quick to train, the single layer networks can then represent a good baseline and a benchmark for comparison with more complex algorithms.

The spectrum of possible decision boundaries of linear networks can be made wider by using the *generalized linear discriminants*:



**Fig. 8.5.** Effectiveness of linear separation surfaces. The left picture shows the XOR problem. No linear decision boundary surface is capable of separating the two classes. On the other hand, a linear surface separating two Gaussian distributions minimizes the error rate in attributing each test sample to the correct distribution.

$$y_k(\mathbf{x}) = \sum_{l=1}^d w_{kl} \phi_l(\mathbf{x}) + w_{k0} \quad (8.20)$$

where the  $\phi_j(\mathbf{x})$  are called *basis functions* and must be chosen appropriately for the problem at hand. As an example, consider the case where  $d = 1$  and the data are then real numbers:

$$y_k(x) = w_{k1} \phi_1(x) + w_{k0}, \quad (8.21)$$

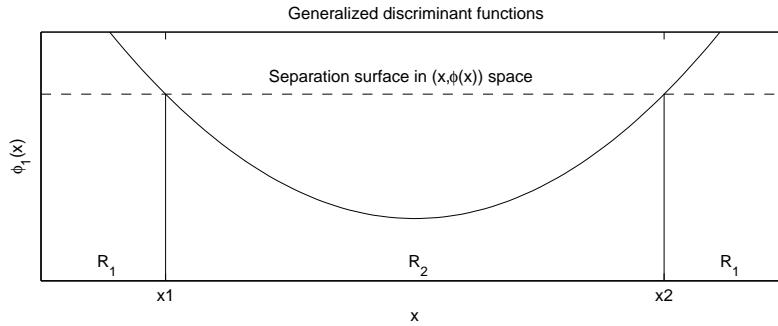
and pose  $\phi_1(x) = a' + bx + cx^2$ . The equation  $y_k(x) > 0$  corresponds then to the following expression:

$$a + bx + cx^2 > 0 \quad (8.22)$$

where  $a = a' + w_{k0}/w_{k1}$ . Consider the case where  $\Delta = b^2 - 4ac > 0$  then the above equation has two distinct real solutions  $x_1$  and  $x_2$ , where  $x_1 < x_2$ , and it can be rewritten as follows:

$$(x - x_1)(x - x_2) > 0 \quad (8.23)$$

which is satisfied in the intervals  $x < x_1$  and  $x > x_2$ . Such a separation surface could not be obtained with simple linear discriminant functions because these can lead only to regions of the form  $x < x_0$ , then can only split the real axis in two parts rather than in three like the generalized function of Equation (8.21). The geometric interpretation of this problem is shown in Figure 8.6: the function  $\phi_1(x)$  maps the points of the real axis onto a parabola in the space  $(x, \phi_1(x))$  and, in such a space, a linear separation surface splits the



**Fig. 8.6.** Generalized linear functions. The picture shows how the function  $\phi(x)$  maps the data points into a parabola in the space  $(x, \phi(x))$ . A linear separation surface in such a space induces a non linear separation surface capable of identifying regions  $R_1$  and  $R_2$  in the original data space  $x$ .

data into three intervals corresponding to  $x < x_1$ ,  $x_1 \leq x \leq x_2$  and  $x > x_2$ . In more general terms, the basis functions represent the data in a space where a linear surface separation corresponds to a more complex surface in the original data space.

One of the earliest examples of single layer networks (if not the earliest one) was based on the generalized discriminant functions approach. The network was called *perceptron* [39] and it was composed of a single processing unit with step activation function (see Section 8.2). At the same time, similar networks called *Adalines* (standing for ADaptive LINear Element) [47] were independently investigated. The perceptron was applied to the problem of recognizing characters and the input data were random pixels extracted from the character images. Since the performance of a single processing unit was too low, the input data were passed through processing elements  $\phi_j$  weighted with adaptable coefficients. The result was the following function:

$$y = g \left( \sum_{i=1}^M w_i \phi_i(\mathbf{x}) + w_0 \right) \quad (8.24)$$

which actually corresponds to a generalized discriminant function given as input to a step function. The limits of the perceptron in addressing problems where linear decision boundaries are not effective stopped the interest in neural networks for around two decades (roughly from the mid sixties to the mid eighties). The availability of computers capable of dealing with more complex network architectures finally made it possible to overcome the perceptron limits by using multilayer neural networks.

## 8.6 Multilayer Networks

This section presents neural networks with more than one layer of connections and, more in particular, the so-called *Multilayer Perceptron* (MLP), a neural network that will be shown to have important properties. Although MLP can have an arbitrary number of hidden layers it has been proven<sup>1</sup>, independently by [9] and [21], that it is adequate one hidden layer for guaranteeing that MLP has *universal approximation property* (or *best approximation property*), i.e. it can approximate arbitrarily well any functional continuous mapping between spaces of finite dimension, provided that the number of hidden neurons (see Figure 8.3) is sufficiently large. In the context of the classification problem, this means that, implicitly, the MLPs can approximate arbitrarily well any decision boundary. This overcomes the main limit of single layer networks that can lead only to linear separation surfaces<sup>2</sup> and explains why in classification and regression tasks no major attention is paid to MLP with more than one hidden layer. This is true only for these tasks but not in general. If we use MLP for feature extraction, e.g. for extracting nonlinear components, three hidden layers are required (see Chapter 11). In the rest of this section we assume that MLP has one hidden layer, i.e. two weights layers and we will show how to train an MLP, i.e. how to find the weights satisfying a predefined criterion over a training set of labeled examples. Section 8.7.4 describes a package enabling one to easily implement, train and test Multilayer networks.

### 8.6.1 The Multilayer Perceptron

The MLP is a feed-forward fully connected network and the corresponding function can be found by simply following the flow of information along the different layers. If the input vectors  $\mathbf{x}$  are  $d$ -dimensional, then the network must have  $d + 1$  input neurons. The input of the extra neuron is always 1 and the weights connecting the extra neuron to the hidden nodes are the biases of these last. The input of the generic node  $j$  in the hidden layer is then:

$$a_j = \sum_{l=1}^d w_{jl}x_l + w_{j0}x_0 = \sum_{l=0}^d w_{jl}x_l \quad (8.25)$$

where  $\mathbf{x} = (x_1, \dots, x_d)$  is the input vector,  $x_0$  is the input of the extra neuron and it is set to 1,  $w_{j0}$  is the bias of hidden node  $j$ , and  $w_{jl}$  ( $l = 1, \dots, d$ ) are the weights of the connections between the input nodes and the hidden node  $j$ .

<sup>1</sup> The result can be obtained using the Stone-Weierstrass theorem [21] or the Hahn-Banach theorem [9].

<sup>2</sup> The generalized linear discriminant functions can actually lead to nonlinear surfaces, but still they cannot approximate any possible decision boundary. See Section 8.5 for more details.



The output  $z_j$  of the  $j^{\text{th}}$  hidden node can be obtained by simply applying the activation function  $\tilde{g}(\cdot)$  of the hidden nodes:

$$z_j = \tilde{g} \left( \sum_{l=0}^d w_{jl} x_l \right), \quad (8.26)$$

where  $j = d + 2, \dots, d + 1 + H$  ( $H$  is the number of hidden nodes), and  $z_{d+1}$  is set to 1 because neuron  $d + 1$  is used to account for the output layer biases. In the same way it is possible to show that the output  $y_k$  of output node  $k$  is:

$$y_k = g \left( \sum_{l=d+1}^{d+1+H} w_{kl} z_l \right) = g \left[ \sum_{j=d+1}^{d+1+H} w_{kj} \tilde{g} \left( \sum_{l=0}^d w_{jl} x_l \right) \right] \quad (8.27)$$

where  $k = d + H + 1, \dots, d + H + O$  ( $O$  is the number of output nodes). Note that when  $g(\cdot)$  is the identity function, the last equation corresponds to the expression of the generalized linear discriminant functions (see Section 8.9).

In general, the activation function of the hidden nodes is nonlinear. The reason is that networks where the hidden nodes have linear activation function are equivalent to networks without hidden nodes [1]. In other words, multilayer networks where the hidden nodes have linear activation function have the same limits as single layer networks (see Section 8.5) and do not have the important properties (see below) of multilayer networks. Linear activation functions in the hidden nodes lead to interesting results only for auto-associative networks, i.e. networks where the target is the input and the number of the hidden neurons is lower than the input dimensionality ( $H < d$ ). In this case, the output of the hidden layer corresponds to a transform of the hidden data known as *principal component analysis* (PCA) which reduces the dimensionality of the data while preserving most of the information they contain (see [2] and Chapter 11 for more details).

When the activation functions are sigmoidal (i.e. logistic sigmoid or hyperbolic tangent) for both hidden and output nodes, then the resulting networks can approximate arbitrarily well any functional continuous mapping from one finite-dimensional space to another if the number of hidden neurons  $H$  is sufficiently large [9]. This result has the important consequence that, in a classification context, any decision boundary surface can be arbitrarily well approximated with an MLP. In other words, while single layer networks lead to a limited range of separation surfaces, multilayer networks can lead to any separation surface. Another important consequence is that when the activation function neurons is a logistic sigmoid, then the MLP can approximate arbitrarily well the a-posteriori probability  $p(\mathcal{C}|\mathbf{x})$  of a class  $\mathcal{C}$  (see Section 8.5.2 for more details).

In order for an MLP to approximate a specific mapping, it is necessary to find the parameter set (i.e. the values of weights and biases) that correspond to such a mapping. This can be done through a training procedure where the

network adapts the parameters based on a set of labeled examples, i.e. pairs  $(\mathbf{x}_k, y_k)$  including an input vector  $\mathbf{x}_k$  and the desired output (the so-called *target*)  $y_k$ . The training algorithm for the MLP's is called *back-propagation* and it is the subject of the next section.

## 8.7 Multilayer Networks Training

As in the cases presented so far in previous chapters, the training procedure is based on the minimization of an error function, or *empirical risk* (see Chapter 5), with respect to the parameter set of the algorithm under examination. In the case of MLPs, the parameter set  $\mathbf{w}$  contains connection weights and neuron biases. The error function is a differentiable function of the network outputs  $y_k$  and these are a function of the network parameters as shown in Equation (8.28), then the error function can be derived with respect to any single parameter in  $\mathbf{w}$ . This enables to minimize the error function by applying different optimization algorithms such as gradient descent. The name *error back-propagation* comes from the fact that the derivation propagates the error from the output nodes to the input nodes [40] (see below for more details).

In general the training algorithms are iterative and each iteration involves two steps that can be considered separately:

- *Evaluation of error function derivatives.* The expression error back-propagation actually refers to this step, although it is used sometimes to define the whole training process. This stage depends on the particular network under examination because the functional expression corresponding to the network, Equation (8.28) in the case of MLP, changes for each architecture.
- *Parameters update.* This stage modifies the network parameters with the goal of minimizing the error function. This stage is independent of the particular network used. In fact, once the derivatives are at disposition, the minimization techniques do not depend any more on the particular network or architecture used.

In the following the two steps are described in more detail.

### 8.7.1 Error Back-Propagation for Feed-Forwards Networks\*

Since the training is supervised, we have a training set which is a collection of input-output patterns, i.e.  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_\ell, \mathbf{t}_\ell)\} \in \mathbb{R}^d \times \mathcal{Y}$ . In the regression problem  $\mathcal{Y}$  is continuous i.e.  $\mathcal{Y} \subseteq \mathbb{R}^O$ . In the classification problem  $\mathcal{Y}$  is discrete, i.e.  $\mathcal{Y} = (\mathcal{C}_1, \dots, \mathcal{C}_O)$ . This representation of the output  $\mathcal{Y}$  is not suitable to be used in a MLP. A more appropriate approach consists in representing  $\mathcal{Y}$  as a discrete subset of  $\mathbb{R}^O$ , i.e.  $\mathcal{Y} = \{+1, -1\}^O$ , where the discrete values +1 and -1 corresponds to the membership and the non-membership

to a given class, respectively. Therefore if the  $m$ -th component of the target  $\hat{\mathbf{y}}$  is  $+1$  then the respective pattern  $\hat{\mathbf{y}}$  belongs to the class  $\mathcal{C}_m$ .

Being said that, the functional form corresponding to a feed-forward network network is:

$$y_k = g \left[ \sum_{j=d+1}^{d+1+H} w_{kj} \tilde{g} \left( \sum_{l=0}^d w_{jl} x_l \right) \right], \quad (8.28)$$

see Equation (8.28), where the biases are included in the summations through extra nodes with input fixed to 1 and do not need to be distinguished from connection weights. The error function has typically the following form:

$$E = \sum_{n=1}^{\ell} \epsilon_n \quad (8.29)$$

where  $\epsilon_n$  is the error, i.e. the *loss function* (see Chapter 5), of the network over the  $n^{\text{th}}$  sample of the training set  $\mathcal{D}$ . The derivative of  $E$  with respect to any parameter  $w_{ij}$  can then be expressed as:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{n=1}^{\ell} \frac{\partial \epsilon_n}{\partial w_{ij}} \quad (8.30)$$

and in the following we can focus on a single  $\partial \epsilon / \partial w_{ij}$  (the index  $n$  is omitted whenever possible).

The derivative of  $\epsilon$  with respect to a weight of the *first layer* can be obtained as follows:

$$\frac{\partial \epsilon}{\partial w_{ij}} = \frac{\partial \epsilon}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} \quad (8.31)$$

where  $a_j$  is the input of node  $i$  in the hidden layer,  $i = d+1, \dots, d+1+H$  and  $j = 0, \dots, d$ . The first term of the above product is called *error* and it is denoted with  $\delta_i$ :

$$\delta_i = \frac{\partial \epsilon}{\partial a_i}. \quad (8.32)$$

Since  $a_i = \sum_{l=0}^d w_{il} x_l$ , the second term of the same product is simply  $x_j$ . As a result, the derivative of  $\epsilon$  with respect to a weight in the first layer can be written as follows:

$$\frac{\partial \epsilon}{\partial w_{ij}} = \delta_i x_j. \quad (8.33)$$

Using the same approach, the derivative of  $\epsilon$  with respect to a weight  $w_{kl}$  in the second layer, i.e.  $k = d+H+1, \dots, d+H+O$  and  $l = d+1, \dots, d+1+H$ , can be written as:

$$\frac{\partial \epsilon}{\partial w_{kl}} = \delta_k z_l. \quad (8.34)$$

where

$$\delta_k = \frac{\partial \epsilon}{\partial z_k} \quad (8.35)$$

The expression of the errors  $\delta_j$  is different for hidden and output nodes. The input nodes are not considered because their activation function is the identity. For the output nodes the error  $\delta_j$  is:

$$\delta_i = \frac{\partial \epsilon}{\partial z_i} = \frac{\partial \epsilon}{\partial y_i} \frac{\partial y_i}{\partial z_i} = g'(z_i) \frac{\partial \epsilon}{\partial y_i}. \quad (8.36)$$

where  $g'(z)$  is simply the first derivative of the activation function of the output nodes  $g(z)$ .

For the hidden nodes we have to take into account the fact that they are connected to all of the output nodes, then it is necessary to sum over all of these:

$$\delta_k = \frac{\partial \epsilon}{\partial a_k} = \sum_{l=1}^O \frac{\partial \epsilon}{\partial z_l} \frac{\partial z_l}{\partial a_k} = \sum_l \delta_l \frac{\partial z_l}{\partial a_k} \quad (8.37)$$

where the expression  $\delta_l$  corresponds to Equation (8.36) because the sum is made over the output neurons. The last missing element is then  $\partial z_l / \partial a_k$  which corresponds to the following expression:

$$\frac{\partial z_l}{\partial a_k} = \frac{\partial}{\partial a_k} \sum_{i=1}^H \tilde{g}(a_i) w_{li} = \tilde{g}'(a_k) w_{lk}. \quad (8.38)$$

By plugging the last expression into Equation (8.37), the result for the hidden nodes errors is:

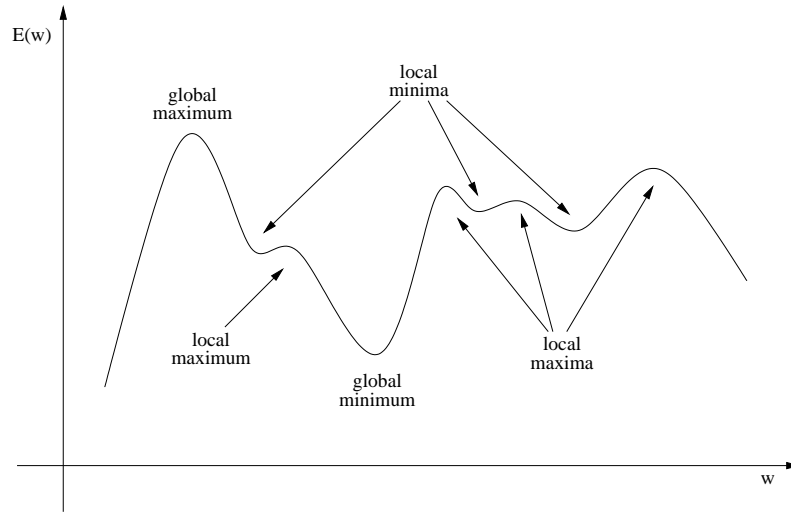
$$\delta_k = \tilde{g}'(a_k) \sum_{l=1}^O w_{lk} g'(z_l) \frac{\partial \epsilon}{\partial y_l}. \quad (8.39)$$

The above results enable one to write the derivative of  $\epsilon_n$  with respect to any network parameter by simply plugging the expression of the activation functions  $g(z)$  and  $\tilde{g}(a)$  as well as of the loss  $\epsilon_n$ . The derivative of  $E$  can then be obtained by simply summing over the errors of all the training set samples.

### 8.7.2 Parameter Update: The Error Surface

The problem of updating the parameters can be thought as the problem of minimizing an error function  $E(\mathbf{w})$ , where  $\mathbf{w}$  is the vector containing all network parameters. The minimization of continuous and differentiable functions of many parameters has been widely studied in the literature and most of the results of such a domain can be applied to the training of neural networks. This section focuses on one of the simplest, but still effective techniques, i.e. *gradient descent*. The reader interested in other methods can find extensive surveys in [1] and, at a tutorial level, in [22].

The error function  $E(\mathbf{w})$  defines a surface, *error surface*, in the parameters space and the goal of the training is to find a point where  $\nabla E = 0$  (see



**Fig. 8.7.** Error surface. The error function defines an *error surface* in the space of the parameters. The goal of the training is to find a minimum of the error surface. Although there is no guarantee that the training leads to the global minimum, the performance in correspondence of local minima is, most of the times, satisfactory.

Figure 8.7). There are several points for which such a property holds. One of them, the so-called *global minimum*, is the point where the error function takes the smallest value. Others are points, called *local minima*, where  $E$  is lower than in the surrounding region but higher than in other regions. Finally, some points where  $\nabla E = 0$  are maxima (local or global) and must be avoided during the training. Due to the high number of parameters, the error surface cannot be explored exhaustively. In general, the training algorithms initialize the parameters with random values  $\mathbf{w}^{(0)}$  and then update them through iterative procedures. At each iteration, the weights are updated as follows:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \Delta\mathbf{w}^{(i)}, \quad (8.40)$$

and different training algorithms correspond to different choices for the update term  $\Delta\mathbf{w}^{(i)}$  (the subscript  $i$  stands for the iteration index). Some algorithms guarantee that  $E(\mathbf{w}^{(i+1)}) \leq E(\mathbf{w}^{(i)})$ , but this still does not guarantee that the error decreases at each iteration. In fact, if the error function falls into a local minimum, there is no way to leave it for a lower local minimum and the algorithm get stuck. Moreover, if  $\mathbf{w}^{(i)}$  corresponds to a relatively flat region of the error surface, the algorithm can evolve very slowly and the training time can become too long.

### 8.7.3 Parameters Update: The Gradient Descent\*

The training is performed using a training set  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_\ell, \mathbf{t}_\ell)\} \subseteq \mathbb{R}^d \times \mathcal{Y}$ , where  $\mathcal{Y} \subseteq \mathbb{R}^O$  (see Section 8.7.1).

There are two possible ways of performing the gradient descent algorithm:

- *On-line learning*: the parameters are updated after each sample of the training set:

$$\Delta \mathbf{w}^{(i)} = -\eta \nabla \epsilon_n |_{\mathbf{w}^{(i)}}, \quad (8.41)$$

where  $\epsilon_n$  denotes the network loss when the input is  $\mathbf{x}_n$ .

- *Off-line learning* (or *Batch learning*): the parameters are updated after that the whole training set has been input to the network:

$$\Delta \mathbf{w}^{(i)} = -\eta \nabla E |_{\mathbf{w}^{(i)}}, \quad (8.42)$$

where  $E = \sum_n \epsilon_n$ .

The parameter  $\eta$  is called *learning rate* and it is one of the main problems of the gradient descent. In fact, if  $\eta$  is too large, the parameters change too much from one iteration to the other and local minima can be missed because the change of position on the error surface is too big. On the other hand, if  $\eta$  is too small, the parameters do not change enough from one iteration to the other, then the network moves too slowly on the error surface and the training time becomes unusefully long. Moreover, the optimal  $\eta$  value is not constant along the training and it should be changed at each iteration.

Equations (8.41) and (8.42) refer to the whole parameter set, but the corresponding expressions can be used for a single parameter by using the results of Section 8.7.1 which shows how to calculate the error function derivatives with respect to any weight or bias. In the on-line version of the gradient descent, the single weights are updated as follows (see Section 8.7.1 for the meaning of symbols):

$$w_{ij}^{(i+1)} = w_{ij}^{(i)} - \eta \frac{\partial \epsilon_n}{\partial w_{ij}} = w_{ij}^{(i)} - \eta \delta_i z_j, \quad (8.43)$$

while in the batch learning, the above expression becomes:

$$w_{ij}^{(i+1)} = w_{ij}^{(i)} - \eta \sum_{n=1}^{\ell} \frac{\partial \epsilon_n}{\partial w_{ij}} = w_{ij}^{(i)} - \eta \sum_{n=1}^{\ell} \delta_i^{(n)} z_j, \quad (8.44)$$

where  $\delta_i^{(n)}$  is the value of  $\delta_i$  for the  $n^{th}$  pattern in the training set.

An important example from the application point of view, is the MLP where hidden nodes have the logistic sigmoid as activation function, output nodes have linear activation function and the loss function is the *quadratic loss* (see Chapter 5), i.e.

$$E = \sum_{i=1}^{\ell} \|\mathbf{y}_i - \mathbf{t}_i\|^2. \quad (8.45)$$

The derivation of the corresponding update rules are left for exercise (see Problem 8.2). The minimization of the error function can be interpreted under the *maximum likelihood principle* (see Chapter 5). In fact, the equation (8.45) can be rewritten as:

$$E = -\ln \exp\left(\sum_{i=1}^{\ell} \|\mathbf{y}_i - \mathbf{t}_i\|^2\right) = -\ln \mathcal{L}(\mathbf{y}, \mathbf{t}). \quad (8.46)$$

Since  $\mathcal{L}(\mathbf{y}, \mathbf{t})$  is the likelihood of the normal joint distribution  $(\mathbf{y}, \mathbf{t})$ , minimizing the error function  $E$  corresponds to assume that the joint distribution  $(\mathbf{y}, \mathbf{t})$  is normal and, at the same time, to maximize its likelihood  $\mathcal{L}(\mathbf{y}, \mathbf{t})$ .

### The Softmax Function

If we assume that the joint distribution  $(\mathbf{y}, \mathbf{t})$  is not normal, the choice of the quadratic loss as loss function is not appropriate. For instance, if we assume that the joint distribution is multinomial, the loss function, using the maximum likelihood principle, is the so-called *cross-entropy* [1], i.e.:

$$\epsilon(\mathbf{y}, \mathbf{t}) = -\sum_{i=1}^O t_i \log y_i. \quad (8.47)$$

Using the cross-entropy as loss function, the error function is:

$$E = -\sum_{i=1}^{\ell} \sum_{l=1}^O t_{il} \log y_{il}, \quad (8.48)$$

where  $t_{il}$  and  $y_{il}$  indicate the  $l$ -th component of  $\mathbf{t}_i$  and  $\mathbf{y}_i$ , respectively.

If we use this error function to train MLP, it is possible to show [1] that the identity activation function on the output nodes, i.e.  $g(z_i) = z_i$  ( $i = 1, \dots, O$ ), has to be replaced with the *softmax function*:

$$g(z_i) = \frac{\exp(z_i)}{\sum_{p=1}^O \exp(z_p)} \quad (i = 1, \dots, O). \quad (8.49)$$

Since  $g(z_i)$  are always positive and their sum is 1, they can be viewed as probabilities. Therefore a MLP having output nodes with the softmax as activation function can be used for probability estimation. The derivation of the corresponding learning rules for a MLP having output nodes with the softmax as activation function is left for exercise.

### 8.7.4 The Torch Package

The *Torch* package<sup>3</sup> is a collection of libraries aimed at the development of several machine learning algorithms [7]. The package enables one to quickly develop, train and test the main kinds of neural networks, including the MLPs described in the previous sections. The library is written in C++, but even a superficial knowledge of such a language is sufficient to use Torch. A tutorial distributed with the code enables one to easily write the programs simulating ANNs.

## 8.8 Learning Vector Quantization

This section will focus on *learning vector quantization* [25] (LVQ), which is a supervised learning algorithm for classification. LVQ is a *prototype-based classifier* that performs a *nearest prototype classification*. Consider a training set  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \subseteq \mathbb{R}^d \times \mathcal{C}$ , where  $y_i$  is a class label that assumes values in  $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_p\}$ . Prototype-based classifiers represent the training set by a set of data points  $\mathcal{M} = (m_1, \dots, m_K) \subseteq \mathbb{R}^d$  in the input space, where  $K \ll \ell$ . The prototypes  $m_i$  are not elements of the training set  $\mathcal{D}$ , but are yielded by the classifier during its phase of learning. A class  $v_i \in \mathcal{C}$  is associated to each prototype  $m_i$  and the classification of a new data point  $\hat{\mathbf{x}}$  is performed assigning the class of the closest prototype. This strategy is called nearest prototype classification. Examples of (unsupervised) prototype-based classifiers are the prototype-based clustering methods, e.g. K-Means and SOM (see Chapter 6), when they are used for classification tasks. LVQ is a supervised prototype method widely used in real time applications like speech [30] and handwriting recognition [5]. We pass to describe the algorithm.

Consider a data set  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \subseteq \mathbb{R}^d \times \mathcal{C}$ .

Using the same terminology introduced in Chapter 6, we call *codebook* the set of data points  $M = \{(m_1, v_1) \dots, (m_K, v_K)\} \subseteq \mathbb{R}^d \times \mathcal{C}$ , where  $K \ll \ell$ . The generic element  $(m_i, v_i) \in M$  is called *codevector*. There are three versions of the LVQ, called LVQ1, LVQ2.1 and LVQ3 respectively. The last two can be considered as successive refinements of the first one. The following shows the three algorithms in detail. The first step of the LVQ1 training is the initialization of the codebook. In general, such a task is performed by randomly selecting  $K$  training samples with the only constraint of having at least one codevector per class. The random selection should be performed by following the a-priori distribution of the labeled examples. In this way, the fraction of codevectors with a certain class  $\mathcal{C}_j$  should roughly correspond to the fraction of training samples with the same class. LVQ1 has the following steps:

---

<sup>3</sup> At the moment of writing this book, software and documentation can be downloaded at the following URL: [www.torch.ch](http://www.torch.ch).



1. Initialize the codebook  $M$ . Fix the number of iterations  $T$ . Set  $t = 1$ .
2. Choose a data point  $(\hat{\mathbf{x}}, \hat{v})$  randomly (with replacement) from the training set  $\mathcal{D}$ .
3. Find the codevector  $\mathbf{m}_c$  such that:

$$\mathbf{m}_c = \arg \min_{i=1, \dots, K} \|\hat{\mathbf{x}} - \mathbf{m}_i\|, \quad (8.50)$$

i.e. the nearest neighbor of  $\hat{\mathbf{x}}$  among the codevectors of  $M$ .

4. Modify the codevector  $\mathbf{m}_c$  into  $\mathbf{m}'_c$  as follows:

$$\mathbf{m}'_c = \left\{ \begin{array}{l} \mathbf{m}_c + \alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_c] \text{ if } \hat{v} = v_c \\ \mathbf{m}_c - \alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_c] \text{ if } \hat{v} \neq v_c. \end{array} \right\}. \quad (8.51)$$

In other words, the codevector  $\mathbf{m}_c$  is moved closer to  $\hat{\mathbf{x}}$  if the two vectors have the same class label and the contrary otherwise. The value of  $\alpha(t)$  must be set empirically (values smaller than 0.1 are advised in [25]) and it decreases linearly with  $t$ .

5. Leave unchanged all codevectors different from  $\mathbf{m}_c$ :

$$\mathbf{m}'_i = \mathbf{m}_i \text{ if } i \neq c. \quad (8.52)$$

Therefore, the only codevector modified is the nearest neighbor of  $\hat{\mathbf{x}}$ , all other codevectors are left unchanged.

6. If  $t < T$  increase  $t$  by one and go to step 2.
7. Return the codebook.

We remark that the updating rule, when the labels of the winning codevector and the input vector are the same, coincides with the learning rule of on-line K-Means. Finally, the termination criterion of LVQ1 can be modified replacing the number of iterations with the achievement of a value of error, a-priori fixed.

LVQ2.1 is a refinement of LVQ1 and is generally carried out after LVQ1. LVQ2.1 has the following steps:

1. Initialize the codebook  $M$  by means of LVQ1. Fix the number of iterations  $T$ . Set  $t = 1$ .
2. Choose a data point  $(\hat{\mathbf{x}}, \hat{v})$  randomly (with replacement) from the training set  $\mathcal{D}$ .
3. Find the codevector  $(\mathbf{m}_i, v_i)$  and  $(\mathbf{m}_j, v_j)$  such that

$$\begin{aligned} \mathbf{m}_i &= \arg \min_{k: v_k = \hat{v}} \|\hat{\mathbf{x}} - \mathbf{m}_k\| \\ \mathbf{m}_j &= \arg \min_{k: v_k \neq \hat{v}} \|\hat{\mathbf{x}} - \mathbf{m}_k\| \end{aligned} \quad (8.53)$$

4. Verify if  $\hat{\mathbf{x}}$  falls in the *window* defined by  $\mathbf{m}_i$  and  $\mathbf{m}_j$ , i.e. if:

$$\frac{1}{s} \leq \frac{\|\hat{\mathbf{x}} - \mathbf{m}_i\|}{\|\hat{\mathbf{x}} - \mathbf{m}_j\|} \leq s \quad (8.54)$$

where  $s = \frac{1+w}{1-w}$  and  $w$  is a constant to be set empirically (values between 0.2 and 0.3 seem to perform well [25]).

5. If  $\hat{\mathbf{x}}$  falls in the window, then the two codevectors are updated as follows:

$$\begin{aligned}\mathbf{m}'_i &= \mathbf{m}_i + \alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_i] \\ \mathbf{m}'_j &= \mathbf{m}_j - \alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_j]\end{aligned}\tag{8.55}$$

see above for  $\alpha(t)$ .

6. If  $t < T$  increase  $t$  by one and go to step 2.

7. Return the codebook.

The goal of LVQ2.1 is to push decision boundaries towards the surface decision yielded by Bayes' rule (see Chapter 5), but no attention is paid to the fact that, the codevectors do not converge to a stable position as  $t$  increases. To prevent this behavior as far as possible, the window  $w$  within the adaptation rule takes place must be chosen carefully. Moreover, the related term

$$\tau = \left| \frac{\|\hat{\mathbf{x}} - \mathbf{m}_j\| - \|\hat{\mathbf{x}} - \mathbf{m}_i\|}{2} \right|,$$

where  $m_i$  and  $m_j$  are defined as in (8.53), yields the hypothesis margin of the classifier [8]. Hence LVQ2.1 can be seen as a classifier which aims at *structural risk minimization* (see Chapter 7) during training, comparable to *support vector machines* (see Chapter 9).

To overcome the LVQ2.1 stability problems, it was necessary to introduce a further correction that tries to deal with this problem. The result is the LVQ3 algorithm which is similar to LVQ2.1. LVQ3 chooses a pattern  $\hat{\mathbf{x}}$  and picks the two closest codevectors  $\mathbf{m}_i$  and  $\mathbf{m}_j$ . If they are in the window and one belongs to the same class of  $\hat{\mathbf{x}}$  and the other not, the LVQ2.1 learning rule is applied. If they are in the window and both codevectors have the same class of  $\hat{\mathbf{x}}$  the following rule is applied:

$$\begin{aligned}\mathbf{m}'_i &= \mathbf{m}_i + \epsilon\alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_i] \\ \mathbf{m}'_j &= \mathbf{m}_j + \epsilon\alpha(t)[\hat{\mathbf{x}} - \mathbf{m}_j].\end{aligned}\tag{8.56}$$

LVQ3 ensures higher stability for the codevectors position as the number of iterations  $t$  increases. The value of  $\epsilon$  must be set empirically and values between 0.1 and 0.5 seem to produce good results [25]. Finally, variants of the LVQ algorithm have been proposed in [17][43].

### 8.8.1 The *LVQ\_PAK* Software Package

The LVQ algorithm described in the previous section is implemented in a package that can be downloaded from the web.<sup>4</sup> This section proposes a quick tutorial (detailed instructions are available in [25]) on the main functions available in the package. The following shows the steps necessary to build a quantizer using a labeled data set and the LVQ1 algorithm:

<sup>4</sup> At the time this book is being written the package is available at the following URL: <http://www.cis.hut.fi/research/som-research/nmrc-programs.shtml>.

1. *Initialization.* The first step is the initialization of the codebook which is performed with the following command:

```
eveninit -noc 200 -din train.dat -cout cbook1.dat -knn 3
```

where `noc` stands for the number of codevectors, `din` corresponds to the name of the file containing the training data, `cout` provides the name of the output codebook file and `knn` verifies that the three nearest neighbors of each initialized codevector have the same label.

2. *LVQ1 training.* The training is performed by the following command:

```
lvq1 -din train.dat -cin cbook1.dat -cout cbook2.dat -rlen
      10000 -alpha 0.05
```

where `cin` stands for the initial codebook (the output of the first step), `rlen` gives the number of training steps (if there are less training samples than training steps, then the same samples are used several times) and `alpha` is the  $\alpha(t)$  parameter.

3. *Test.* The effectiveness of the codebook can be measured with the following command:

```
accuracy -din test.dat -cin cbook2.dat
```

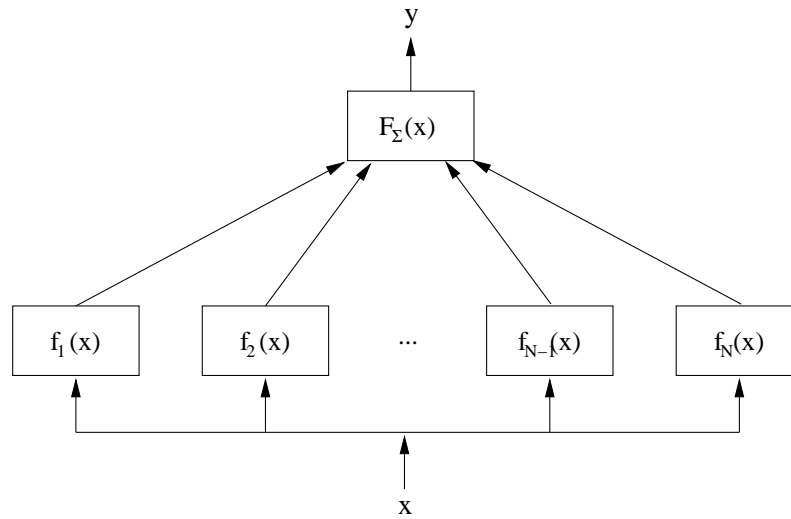
where `test.dat` is a file containing labeled test data (different from the data in `training.dat`). The accuracy is measured in terms of *recognition rate*, i.e. number of input vectors mapped into the correct label.

The LVQ\_PAK offers several more functions and options which enable one to obtain quantizers corresponding to the algorithms shown in the previous section.

## 8.9 Ensemble Methods

This section presents the *ensemble* methods, i.e. the techniques aimed at combining the predictions of a set of single *learners*, e.g. a set of classifiers or a set of regressors,  $f_i(\mathbf{x})$ , trained individually, in order to obtain an overall learner  $F_\Sigma(\mathbf{x})$  which performs better than any single  $f_i(\mathbf{x})$  (see Figure 8.8). In this section we will focus on ensemble methods for classification.

The combination of the single output can be performed in different ways (see [23] for a survey), but commonly it consists of a majority vote (i.e. the output of  $F_\Sigma$  is the most frequent output among the values of the  $f_i(\mathbf{x})$ ), or of the average of the  $f_i(\mathbf{x})$  output values. The set  $F = \{f_1(\mathbf{x}), \dots, f_N(\mathbf{x})\}$  is called *classifier ensemble* and it can be obtained with different techniques (see below). This subject is explored in detail in both monographies [26] and tutorials [10][11][36]. The rest of this part will show some possible reasons of the ensemble improvements over single classifiers (Section 8.9.1) and the main techniques for creating ensembles (Section 8.9.2).



**Fig. 8.8.** Classifiers ensemble. The same input is presented to different classifiers and their output is combined resulting into a classifier  $F_\Sigma(\mathbf{x})$ .

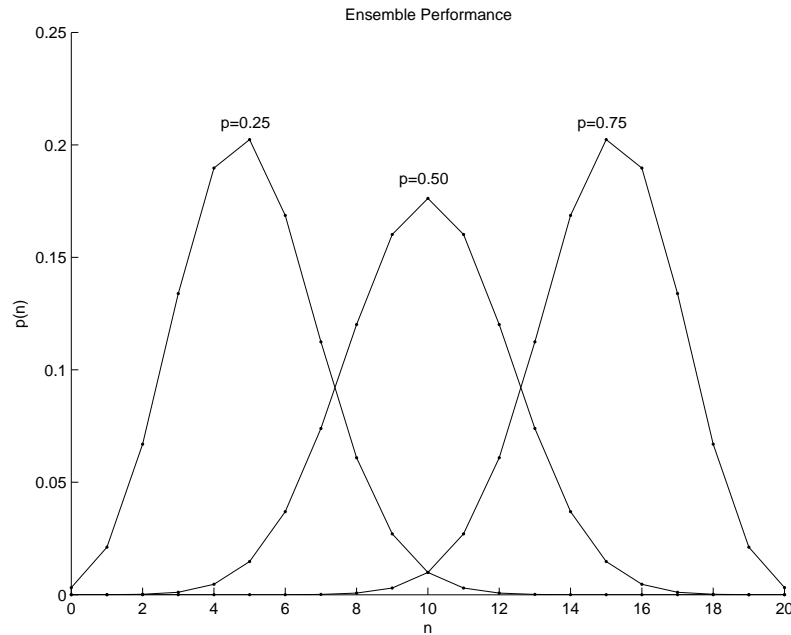
### 8.9.1 Classifier Diversity and Ensemble Performance\*

Classifier combination is an operation that makes sense only if the classifiers are *diverse*, i.e. if they make different errors on the same data [38] or, in more rigorous terms, are statistically independent. In fact, given an ensemble of  $N$  classifiers, it is reasonable to expect that those who misclassify a given input  $\mathbf{x}$  distribute their output more or less uniformly over the wrong labels, while those who classify correctly the same  $\mathbf{x}$  provide the same output, i.e. the correct class. In this way, a simple majority vote can lead  $F_\Sigma$  to assign the correct label to  $\mathbf{x}$ .

As an example, consider an ensemble of  $N$  classifiers with recognition rate  $p$ , where the recognition rate is the percentage of correctly classified samples. If the outputs of the classifiers are statistically independent, then the probability of  $n$  classifiers giving the right answer is:

$$p(n) = p^n(1-p)^{N-n} \binom{N}{n} = p^{N-n}(1-p)^n \frac{N!}{n!(N-n)!} \quad (8.57)$$

and it is plotted as a function of  $n$  in Figure 8.9 for  $N = 20$  and  $p \in \{0.25, 0.50, 0.75\}$ . The plot shows that the most probable number of classifiers giving the right answer is 5, 10 and 15 for the three values of  $p$  respectively. Although shown for a specific case, this corresponds to a general result: when the recognition rate is higher than 0.5, the most probable  $n$  is always higher than  $N/2$ . It is even more important to note that the probability of  $n$  being higher than  $N/2$  is 0.004, 0.41 and 0.98 for  $p = 0.25$ ,  $p = 0.5$  and  $p = 0.75$



**Fig. 8.9.** Number of correct classifiers. The plots show the probability of  $n$  classifiers providing the correct answer for  $p = 0.25$ ,  $0.5$  and  $0.75$  respectively.

respectively (in the case of the above example). In other words, the application of a simple majority vote<sup>5</sup> leads to the correct result in a percentage of cases higher than the recognition rate of the single classifiers. It is worth to remember that such a result applies only when the output of the classifiers is statistically independent.

The same phenomenon can be seen under a different perspective [38]. A classifier  $f_i(\mathbf{x})$  can be seen as an approximation of a true (and unknown) function  $f(\mathbf{x})$ . In general, each classifier is trained to minimize the *empirical risk*:

$$MSE[f_i] = \frac{1}{\ell} \sum_{j=1}^{\ell} (y_j - f_i(\mathbf{x}_j))^2 \quad (8.58)$$

where  $\ell$  is the number of training samples and the *quadratic loss* (see Chapter 5) is chosen as loss function.

The above expression can be thought of as the average squared value of  $m_i(\mathbf{x}) = \mathbf{y}_l - f_i(\mathbf{x}_l)$ :

$$MSE[f_i] = \mathcal{E}[m_i^2(\mathbf{x})]. \quad (8.59)$$

<sup>5</sup> The expression *majority vote* means that the output of  $F_{\Sigma}(\mathbf{x})$  is the most frequent output of the single ensemble classifiers  $f_i(\mathbf{x})$ .

Consider now the ensemble obtained by simply averaging over the output of the single classifiers:

$$F_{\Sigma}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}), \quad (8.60)$$

by plugging Equation 8.58 into the last expression we have:

$$F_{\Sigma}(\mathbf{x}) = f(\mathbf{x}) - \frac{1}{N} \sum_{i=1}^N m_i(\mathbf{x}). \quad (8.61)$$

If the  $m_i(\mathbf{x})$  are mutually independent with zero mean, then the *MSE* of the ensemble is as follows:

$$MSE[F_{\Sigma}(\mathbf{x})] = \mathcal{E}\left[\left(\frac{1}{N} \sum_{i=1}^N m_i(\mathbf{x})\right)^2\right] = \frac{1}{N^2} \mathcal{E}\left[\sum_{i=1}^N m_i^2(\mathbf{x})\right] \quad (8.62)$$

(the demonstration is the subject of Problem 8.6), and it corresponds to:

$$MSE[F_{\Sigma}(\mathbf{x})] = \frac{1}{N} \sum_{i=1}^N MSE[f_i(\mathbf{x})], \quad (8.63)$$

i.e. the average of the empirical mean squared errors of the different classifiers  $f_i(\mathbf{x})$ . Such a result shows that, in principle, the MSE can be arbitrarily decreased by simply increasing  $N$ . On the other hand, in practice the independence assumptions made to obtain the above equation are less and less verified when the number of classifier increases [38]. In fact, both empirical and theoretical investigations show that performance of an ensemble improves up to 20-25 classifiers [36] and then it saturates.

This section has shown that the diversity is a key factor for the classifier ensembles. The next section shows what are the main methods to create ensembles of classifiers as diverse as possible.

### 8.9.2 Creating Ensemble of Diverse Classifiers

This section proposes a quick survey of the most common methods used to build ensembles of classifiers as diverse as possible.

#### Bayesian Voting

Consider a training set  $\mathcal{X}$  and a classifier  $f(\mathbf{x})$  which can be trained. The result of the training is a hypothesis  $h(\mathbf{x})$ , i.e. a particular instance of the classifier determined by a specific parameter set. As an example, consider the neural networks introduced at the beginning of this chapter, a network with a given architecture (number of nodes and structure of the connections) and

a given set of weights  $W$  corresponds to a hypothesis  $h(\mathbf{x})$ . The set of all possible networks with the same architecture, but different parameters sets is called *Hypothesis Space*  $\mathcal{H}$ . Each neural network is an element of  $\mathcal{H}$  and each element of  $\mathcal{H}$  is a neural network with a given architecture.

Consider the conditional probability distribution  $p(f(\mathbf{x}) = \mathbf{y}|h, \mathbf{x})$ , i.e. the probability of observing the output  $\mathbf{y}$  given the hypothesis  $h$  and the input  $\mathbf{x}$ . The problem of predicting the value of  $f(\mathbf{x})$  can be thought of as the problem of estimating  $p(f(\mathbf{x}) = \mathbf{y}|\mathcal{X}, \mathbf{x})$ . Such a probability can be rewritten as a weighted sum:

$$p(f(\mathbf{x}) = \mathbf{y}|\mathcal{X}, \mathbf{x}) = \sum_{h \in \mathcal{H}} h(\mathbf{x})p(h|\mathcal{X}), \quad (8.64)$$

i.e. as an ensemble where each classifier is weighted following its posterior probability  $p(h|\mathcal{X})$ . The posterior can be estimated with the product  $p(\mathcal{X}|h)p(h)$  (keeping into account that  $p(\mathcal{X})$  is a constant).

Such an approach has two main problems. The first is that  $p(h)$  is not often known and it is typically selected based on computational convenience rather than on an actual knowledge of the hypothesis distribution. Moreover, while for some classifiers the hypothesis space can be enumerated, for others (e.g. neural networks or support vector machines) it can only be sampled.

### Bagging

One of the most straightforward ways to obtain diversity is to train the same classifier over different training samples. Such an approach is especially suitable for algorithms that are heavily affected by changes even small in the training set [10].

The simplest method in this family of approaches is the *Bootstrap Aggregation* [3], often called *Bagging*. Bagging is derived by a statistical method called *bootstrap* [14].

Given a training set  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$ , the bootstrap method consists in creating independently  $M$  new data sets  $\mathcal{D}_1, \dots, \mathcal{D}_M$ . Each data set  $\mathcal{D}_i$  is generated by randomizing picking  $\ell$  data points from  $\mathcal{D}$ , with replacement. Therefore some duplicated data points can exist in  $\mathcal{D}_i$ .

Being said that, in bagging the same learning algorithm is presented to  $M$  different training sets obtained by randomly drawing  $n < \ell$  data points from the original training set  $\mathcal{D}$ , with replacement. A common choice consists in choosing the cardinality  $n$  of each data subset (the *bootstrap aggregate*) equal to  $\sim \frac{2\ell}{3}$ . Each bootstrap aggregate is used to train a classifier, by means of the same learning algorithm. Finally, the classification is produced by means of a majority vote on the  $M$  classifiers. The properties of bagging have been widely explored. In particular bagging seems to have stability properties. A learning algorithm is called *unstable*<sup>6</sup> if small changes in the training data produces different classifiers and very large changes in their performances (e.g.

<sup>6</sup> Examples of unstable classifiers are the decision trees classifiers, which are not discussed in the book.

recognition rate). Bagging averages over the eventual discontinuities that can occur in a classifier, generated by the presence or the absence of a given pattern, making the classifier more stable. Finally, we remark that bagging is an example of a statistical method called *arcing*, acronym of *adaptive reweighting and combining* [4]. Arcing indicates the reusing data in order to improve classification.

Another method based on the majority voting consists in obtaining  $M$  classifiers by alternatively dropping out of the training set  $M$  randomly extracted disjoint subsets. Such a method is similar to an  $M$ -fold crossvalidation and the ensembles obtained in this way are often called *crossvalidated committees* [37].

## Boosting

The last ensemble method based on training resampling is the so-called *boosting* [44]. We describe the boosting method considering a binary classification problem, i.e. each data point can only be classified in two different ways,  $\mathcal{C}_1$  and  $\mathcal{C}_2$ . Given a training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\} \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2\}$ , we consider three different classifiers  $F_1$ ,  $F_2$  and  $F_3$ . First we create a data set  $\mathcal{D}_1$  randomly picking  $n < \ell$  data points from the training set  $\mathcal{D}$  without replacement. Then we train the first classifier  $F_1$  with  $\mathcal{D}_1$ . The classifier  $F_1$  is a *weak learner*, namely its performances are slightly better than the coin toss. The next step of the boosting consists in creating a new data set  $\mathcal{D}_2$  generated as follows. We make a coin toss. If the result is heads we present, one by one, the data points of  $\mathcal{D}$  which does not belong to  $\mathcal{D}_1$  until the classifier  $F_1$  misclassifies a data sample. We add this pattern to  $\mathcal{D}_2$ . We repeat the coin toss. If the result is heads we look again for another misclassified pattern by  $F_1$  and we add it to  $\mathcal{D}_2$ . If the result is tails we look for a data point that  $F_1$  classifies correctly and we add this pattern to  $\mathcal{D}_2$ . We repeat the procedure until no pattern can be added to  $\mathcal{D}_2$ . In this way, the data set  $\mathcal{D}_2$  contains half of the pattern correctly classified whereas the other half is formed by pattern misclassified by the classifier  $F_1$ . We train the second classifier  $F_2$  on  $\mathcal{D}_2$ . Then we look for a third data set  $\mathcal{D}_3$  generated as follows. We present the remaining data points of  $\mathcal{D}$ , i.e. the patterns of  $\mathcal{D}$  that are neither elements of  $\mathcal{D}_1$  nor elements of  $\mathcal{D}_2$ , to the classifiers  $F_1$  and  $F_2$ . If the classifiers do not agree we add the data point to  $\mathcal{D}_3$ , otherwise the pattern is discarded. We repeat the procedure until it is not possible to add pattern to  $\mathcal{D}_3$ . Then we train the last classifier  $F_3$  on  $\mathcal{D}_3$ . Finally, a new test pattern  $\hat{\mathbf{x}}$ , that does not belong to  $\mathcal{D}$ , is classified on the basis of the responses of the three classifiers. If the classifiers  $F_1$  and  $F_2$  agree about the class to assign to  $\hat{\mathbf{x}}$ , the class is assigned to  $\hat{\mathbf{x}}$ . Otherwise, we assign to  $\hat{\mathbf{x}}$ , the class assigned by  $F_3$ . We conclude this description of boosting remarking that the cardinality of the first data set is usually chosen equal to  $n = \frac{\ell}{3}$ .



## AdaBoost

Among the variants of boosting, the most popular is *AdaBoost* [15]. AdaBoost, acronym of *adaptive boosting*, allows to add weak learner until a training error, apriori fixed, is achieved. In AdaBoost algorithm a weight  $W$  is associated to each pattern of the training set.  $W$  represents the pattern probability to be chosen by a component classifier of the ensemble. If the pattern is correctly classified  $W$  is decreased, otherwise it is increased. Therefore this algorithm pays particular attention to the pattern difficult to be classified. Let  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2\}$  be a training set and  $\mathcal{W} = \{W(1), \dots, W(\ell)\}$  where  $W_i$  is the weight associated to the generic pattern  $(\mathbf{x}_i, y_i)$ . AdaBoost algorithm has the following steps:

1. Initialize  $k = 0$ ,  $K_M$ ,  $W_1(i) = \frac{1}{\ell}$  ( $i = 1, \dots, \ell$ )
2.  $k = k + 1$
3. Train classifier  $F_k$  on  $\mathcal{D}$  using  $W_1(i)$
4. Compute the loss function  $L_k$  of  $F_k$
5. Compute

$$\alpha_k = \frac{1}{2} \ln \frac{1 - F_k}{F_k}.$$

6. Compute

$$W_{k+1}(i) = \begin{cases} A_k W_k(i) \exp(-\alpha_k) & \text{if } h_k(\mathbf{x}_i) = y_i \\ A_k W_k(i) \exp(\alpha_k) & \text{if } h_k(\mathbf{x}_i) \neq y_i \end{cases}$$

where  $A_k$  is such that  $\sum_{i=1}^{\ell} W_k(i) = 1$  and  $h_k(\mathbf{x}_i)$  represents the class associated to  $\mathbf{x}_i$  by the classifier  $F_k$ .

7. if  $k < K_M$  go to step 2
8. return  $E_k$  and  $\alpha_k$  ( $k = 1, \dots, K_M$ )

To classify a new test point  $\hat{\mathbf{x}}$ , AdaBoost computes a weighted sums of the outputs (or hypotheses)  $h_k(\hat{\mathbf{x}})$  by the classifier  $F_k$ :

$$\mathcal{F}(\hat{\mathbf{x}}) = \sum_{k=1}^{K_M} \alpha_k h_k(\hat{\mathbf{x}}). \quad (8.65)$$

In the case of binary classification, the decision rule is given by  $\text{sgn}(\mathcal{F}(\hat{\mathbf{x}}))$ , where  $\text{sgn}(\cdot)$  is the signum function.

Finally, we remark that Adaboost algorithm with some *ad hoc* modifications can be applied to regression problems [16].

## Feature-Based Methods

When the input vectors  $\mathbf{x}$  contain a high number of redundant features, the diversity can be obtained by using different feature subsets to train the ensemble classifiers. The literature reports few examples of such a technique [6][45]

and the results seem to suggest that cannot be applied for small feature sets. In fact, in such a case the removal of certain features can lead to classifiers with a recognition rate below 50% (see Section 8.9.1 for the consequences).

### Target-Based Methods

The labels of the training samples are a further source of diversity. A method called *error-correcting output code* [12] splits the data classes into two groups  $A_l$  and  $B_l$  and builds a binary classifier  $h_l(\mathbf{x})$  capable of assigning an input vector to one of the two class groups. The process is repeated  $L$  times resulting into an ensemble of classifiers. Each time a classifier  $h_l(\mathbf{x})$  assigns an input vector to a class group, then all the classes into such group receive one vote. Once the output of all  $h_l(\mathbf{x})$  classifiers is available, the class that has received the highest number of votes is taken as output of the ensemble.

## 8.10 Conclusions

In this chapter we have described the most popular supervised neural network, the Multilayer Perceptron. We have presented a Learning Vector Quantization, which is a prototype-based classifier method quite effective in real time applications. We also review ensemble methods focusing on the ones for the classification task. Finally, we provide some bibliographical remarks. A fundamental work, for its historical value, on neural networks is [42]. Multilayer Perceptron is discussed in detail in [13][18][20][33]. A milestone in the literature on MLP is [1]. Backpropagation was historically introduced in [46] but it was fully discussed in [41]. Learning vector quantization is discussed in [24]. A bibliography on learning vector quantization can be found in [35]. Finally, a comprehensive survey of the ensemble methods is [26], where an entire monography is devoted to the topic.

## Problems

**8.1.** Show that for the LDF corresponding to Equation (8.10), the distance of a point with respect to the surface  $y(\mathbf{x}) = 0$  is  $y(\mathbf{x})/\|\mathbf{w}\|$ .

**8.2.** Find the on-line gradient descent update rules for an MLP where hidden nodes have the logistic sigmoid as activation function, the output nodes have a linear activation function and the loss function is the quadratic loss (see [1] for the solution).

**8.3.** Find the on-line gradient descent update rules for an MLP when the loss function is the cross-entropy.

**8.4.** Using the maximum likelihood principle, prove that if the joint distribution  $(\mathbf{y}, \mathbf{t})$  is multinomial then the loss function is the cross-entropy.

**8.5.** Use the Torch package ([www.torch.ch](http://www.torch.ch)) to implement, train and test a multilayer perceptron. If you have no data at disposition, you can find several interesting benchmarks at the following URL:

<http://www.ics.uci.edu/~mllearn/MLRepository.html>

**8.6.** Demonstrate that, if the  $m_i(\mathbf{x})$  are statistically independent, then:

$$\mathcal{E}\left[\left(\frac{1}{N} \sum_{i=1}^N m_i(\mathbf{x})\right)^2\right] = \frac{1}{N^2} \mathcal{E}\left[\sum_{i=1}^N m_i^2(\mathbf{x})\right] \quad (8.66)$$

(see Appendix A for help).

**8.7.** Use the *LVQ\_PAK* package to classify the same data used in Problem 8.5. Compare the results obtained by the two classifiers. Do the classifiers perform different errors? What is the percentage of cases where both classifiers are correct? And what the percentage of cases where only one of the two classifiers is wrong?

**8.8.** Train an MLP using different initializations for the weights. Use the resulting networks to build an ensemble and measure the improvement with respect to the best and the worse single MLP (for the data see Problem 8.5).

**8.9.** Consider the *Iris Plant* data set that can be found in the repository introduced in problem 8.5. The data set contains 150 four dimensional samples belonging to three different classes. Implement and train an autoassociative MLP (i.e. an MLP that has the same vector as input and output) with two hidden nodes and, after the training, plot the output of the hidden nodes in a two dimensional scatter-plot. Can you still observe the clusters corresponding to the three classes? If you use the output of the hidden nodes as input to a classifier, do you obtain the same classification performance as when you use the original four dimensional vectors?

**8.10.** Create an ensemble of neural networks using the *Error-correcting output code* approach (see Section 8.9.2).



---

## References

1. C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.
2. H. Bourlard and Y. Kamp. Auto-association by Multi-Layer Perceptron and Singular Value Decomposition. *Biological Cybernetics*, 59:291–294, 1988.
3. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
4. L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–824, 1998.
5. F. Camastra and A. Vinciarelli. Cursive character recognition by learning vector quantization. *Pattern Recognition Letters*, 22(6-7):625–629, 2001.
6. K.J. Cherkauer. Human expert-level performance on a scientific image analysis task by a system using combined Artificial Neural Networks. In *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 15–21, 1996.
7. R. Collobert, S. Bengio, and J. Mariethoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR-02-46, IDIAP Research Institute, 2002.
8. K. Crammer, R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin analysis of the LVQ algorithm. In *Advances in Neural Information Processing Systems*, volume 14, pages 109–114, 2002.
9. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
10. T. Dietterich. Ensemble methods in machine learning. In *Proceedings of 1<sup>st</sup> International Workshop on Multiple Classifier Systems*, pages 1–15, 2000.
11. T.G. Dietterich. Ensemble learning. In M. Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press, 2002.
12. T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
13. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
14. B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.
15. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *International Conference in Machine Learning*, pages 138–146, 1996.

16. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
17. B. Hammer and T. Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9):1059–1068, 2002.
18. S. Haykin. *Neural Networks: a comprehensive foundation*. Prentice-Hall, 1998.
19. R. Hecht-Nielsen, editor. *Neurocomputing*. Addison-Wesley, 1990.
20. J. Hertz, A. Krogh, and R.G. Palmer, editors. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.
21. K. Hornik, M. Stinchcombe, and H. White. Multi-Layer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
22. A.K. Jain, J. Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *IEEE Computer*, pages 31–44, 1996.
23. J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.
24. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 1997.
25. T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. Lvsq\_pak: the Learning Vector Quantization program package. Technical Report A30, Helsinki University of Technology - Laboratory of Computer and Information Science, 1996.
26. L. Kuncheva. *Combining Pattern Classifiers*. Wiley-Interscience, 2004.
27. J.L. McClelland, G.E. Hinton, and D.E. Rumelhart. A general framework for parallel distributed processing. In J.L. McClelland and Rumelhart, editors, *Parallel Distributed Processing*, volume Vol. 1: Foundations, pages 45–76. MIT Press, 1986.
28. J.L. McClelland, D.E. Rumelhart, and G.E. Hinton. The appeal of a parallel distributed processing. In J.L. McClelland and Rumelhart, editors, *Parallel Distributed Processing*, volume Vol. 1: Foundations, pages 3–44. MIT Press, 1986.
29. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 9:127–147, 1943.
30. E. McDermott and S. Katagiri. Prototype-based minimum classification error/generalized probabilistic descent training for various speech units. *Computer Speech and Languages*, 8(4):351–368, 1994.
31. D.A. Medler. A brief history of connectionism. *Neural Computing Surveys*, 1:61–101, 1998.
32. M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, 1969.
33. T. Mitchell. *Machine Learning*. Mc Graw-Hill, 1997.
34. J. Nolte. *The human brain: an introduction to its functional anatomy*. Mosby, 2002.
35. M. Oja, S. Karski, and T. Kohonen. Bibliography of self-organizing map papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156, 2002.
36. D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.
37. B. Parmanto, P.W. Munro, and H.R. Doyle. Improving committee diagnosis with resampling techniques. In *Advances in Neural Information Processing Systems*, volume 8, pages 882–888, 1996.

38. M.P. Perrone and L.N. Cooper. When networks disagree: ensemble methods for hybrid neural networks. In R.J. Mammone, editor, *Artificial Neural Networks for speech and vision*, pages 126–142. Chapman & Hall, 1993.
39. F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of vrain mechanisms*. Spartan, 1961.
40. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In J.L. McClelland and Rumelhart, editors, *Parallel Distributed Processing*, volume Vol. 1: Foundations, pages 318–362. MIT Press, 1986.
41. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
42. D.E. Rumelhart and J.L. McClelland, editors. *Parallel Distributed Processing*. MIT Press, 1986.
43. A. S. Sato and K. Yamada. Generalized learning vector quantization. In *Advances in Neural Information Processing Systems*, volume 7, pages 423–429, 1995.
44. R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
45. K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–404, 1996.
46. P. J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Technical report, Harvard University, Ph. D. Dissertation, 1974.
47. B. Widrow and M.E. Hoff. Adaptive switching circuits. In *Convention Record of the Institute of Radio Engineers, Western Electronic Show and Convention*, pages 96–104. Institute for Radio Engineers, 1960.





## Kernel Methods

### What the reader should know to understand this chapter

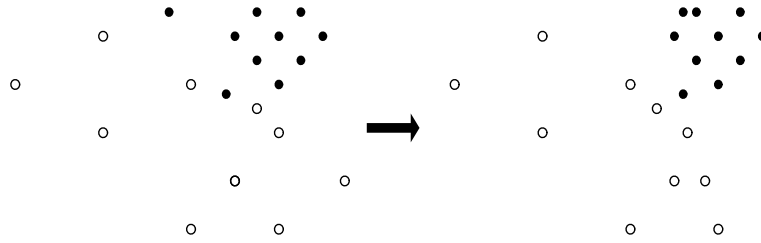
- Notions of calculus.
- Chapters 5, 6, and 7.
- Although the reading of Appendix D is not mandatory, it represents an advantage for the chapter understanding.

### What the reader should know after reading this chapter

- Support vector machines for classification and regression.
- Gaussian Processes.
- Kernel PCA.
- Kernel fisher discriminant.
- One class SVM.
- Kernel and spectral methods for clustering.

## 9.1 Introduction

Kernel methods are algorithms which allow to project implicitly the data in a high-dimensional space. The use of kernel functions to make computations was introduced by [1] in 1964. Two decades later several authors [60][68][70] proposed a neural network, *radial basis function (RBF)*, based on the kernel functions which was widely used in many applicative fields. Since 1995 kernel methods have conquered a fundamental place in machine learning when *support vector machines (SVMs)* were proposed. In several applications, SVMs have showed better performances in comparison with other machine learning algorithms. SVM strategy can be summarized in two steps. In the first step the data are projected implicitly onto a high-dimensional space by means of the *kernel trick* [74] which consists of replacing the inner product between data vectors with a kernel function. The second step consists of applying a linear



**Fig. 9.1.** Data in the input space (at left of the arrow) and their projections in a new space (at right of the arrow).

classifier to the projected data. Since a linear classifier can solve a very limited class of problems, the kernel trick is used to empower the linear classifier, making SVM capable of solving a larger class of problems.

The enormous success of SVMs has induced the researchers to extend the SVM strategy to other existing algorithms, i.e. using the kernel trick to empower learning algorithms, already present in the literature, improving their performances. Therefore with the term *kernel methods* we generally indicate algorithms that use the kernel trick. The basic idea of kernel methods consists in looking for an appropriate mapping of data such that it is easier to process the projected data. To illustrate this concept, we consider Figure 9.1. The data in the input space are not linearly separable (see Chapter 7), i.e. there does not exist a line<sup>1</sup> that separates black disks from white circles. However, if we choose an appropriate mapping then the data projections are linearly separable and can be processed by a linear classifier (e.g. a linear discriminant).

The aim of this chapter is to propose an overview of the main kernel methods, neglecting, for sake of space, those algorithms, like the radial basis function, which are not popular in machine learning community anymore. The chapter is organized as follows: Section 9.2 describes the basic tools of the optimization theory used in the kernel methods. Sections 9.3 and 9.4 are devoted to support vector machines for classification. Section 9.5 introduces Support Vector Machines for Regression. Section 9.6 describes Gaussian processes exploring their connections with support vector machines. Sections 9.7 and 9.8 present respectively the kernel Fisher discriminant and the kernel PCA. Section 9.9 discusses the support vector machine, the so-called one-class SVM, when the data are only formed by positive examples. Section 9.10 is devoted to kernel and the spectral method for clustering. Section 9.11 reviews the main public domain software packages that implement kernel methods. Finally, in Section 9.12 some conclusions are drawn.

<sup>1</sup> If the input dimensionality is higher than 2, the line has to be replaced with a plane or a hyperplane.

## 9.2 Lagrange Method and Kuhn Tucker Theorem

In this section we describe the basic tools of the optimization theory used in the construction of the kernel methods. The first method for solving optimization problems, the *Fermat optimization theorem*, was discovered in 1629 and published 50 years later [26]. The Fermat optimization theorem provides a method for finding the minimum or the maximum of functions defined in the entire space, without constraints. We only state the theorem, omitting the proof for the sake of brevity.

**Theorem 14 (Fermat)** *Let  $f$  be a function of  $n$  variables differentiable at the point  $x^*$ . If  $x^*$  is a point of local extremum of the function  $f(x)$ , then the differential of the function in the point  $x^*$   $Df(x^*)$  is*

$$Df(x^*) = 0, \quad (9.1)$$

which implies

$$\frac{\partial f(x^*)}{\partial x_1} = \frac{\partial f(x^*)}{\partial x_2} = \dots = \frac{\partial f(x^*)}{\partial x_n} = 0. \quad (9.2)$$

A point for which Equation (9.1) holds is called a *stationary point*. Fermat optimization theorem provides a method for finding the stationary points of functions. The method consists in solving the system (9.2) of  $n$  equations with  $n$  unknown values  $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ .

### 9.2.1 Lagrange Multipliers Method

The next step in the optimization theory was done by [49] in 1788 who provides a method for solving the optimization problem with constraints (*conditional optimization problem*). The conditional optimization problem consists in minimizing (or maximizing) the function  $f, f : \mathbb{R}^n \rightarrow \mathbb{R}$  under  $m$  constraints

$$g_1(x) = g_2(x) = \dots = g_m(x) = 0. \quad (9.3)$$

We consider only functions  $g_r, r = 1, \dots, m$  that possess some differentiability properties. We assume that in the subset  $X$  of the space  $\mathbb{R}^n$  all functions  $g_r$  and their partial derivatives are continuous. We have the following definition:

**Definition 20** *Let  $X \subseteq \mathbb{R}^n$  be and  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . We say that  $x^* \in X$  is a point of local minimum in the problem of minimizing  $f$  under constraints (9.3) if there exists  $\epsilon > 0$  such that  $\forall x$  that satisfy (9.3) and*

$$\|x - x^*\| < \epsilon \quad (9.4)$$

the inequality

$$f(x) \geq f(x^*) \quad (9.5)$$

holds.

The definition of maximum is analogous.

Now we pass to define the function  $\mathbb{L}$  (*Lagrangian*), as follows:

$$\mathbb{L}(x, \lambda, \lambda_0) = \lambda_0 f(x) + \sum_{k=1}^m \lambda_k g_k(x), \quad (9.6)$$

where the real values  $\lambda_0, \lambda_1, \dots, \lambda_m$  are called *Lagrange multipliers*. The following theorem was proven by [49], whose proof is omitted for the sake of the brevity.

**Theorem 15 (Lagrange)** *Let the functions  $g_k(x)$ ,  $k = 0, 1, \dots, m$  be continuous and differentiable in a vicinity of  $x^*$ . If  $x^*$  is the point of a local extremum, then one can find Lagrange multipliers  $\lambda^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)$  and  $\lambda_0^*$  which are not equal to zero simultaneously such that the differential of the Lagrangian  $D\mathbb{L}(x^*, \lambda^*, \lambda_0^*)$  is null (**stationary condition**), i.e.*

$$D\mathbb{L}(x^*, \lambda^*, \lambda_0^*) = 0. \quad (9.7)$$

That implies

$$\frac{\partial \mathbb{L}(x^*, \lambda^*, \lambda_0^*)}{\partial x_i} = 0 \quad i = 1, 2, \dots, n. \quad (9.8)$$

To guarantee that  $\lambda_0 \neq 0$  it is sufficient that the  $m$  vectors  $Dg_1(x^*), Dg_2(x^*), \dots, Dg_m(x^*)$  are linearly independent. Where  $Dg_i(x^*)$  stands, respectively, for the differential of  $g_i(x^*)$  ( $i=1, \dots, m$ ).

Therefore to find the stationary point  $x^*$  the system formed by the following  $n + m$  equations

$$\frac{\partial}{\partial x_i} \left( \lambda_0 f(x) + \sum_{k=1}^m \lambda_k g_k(x) \right) = 0 \quad (i = 1, \dots, n) \quad (9.9)$$

$$g_1(x) = g_2(x) = \dots = g_m(x) = 0 \quad (9.10)$$

must be solved.

The system has  $n + m$  equations with  $n + m + 1$  unknown values. Therefore the system is *indeterminate*, i.e. has infinite solutions.<sup>2</sup> However Lagrange multipliers are defined with accuracy up to a common multiplier.

If  $\lambda_0 \neq 0$  then one can multiply all Lagrange multipliers by a constant to obtain  $\lambda_0 = 1$ . Hence the number of equations becomes equal to the number of unknowns. The system assumes the final form:

$$\frac{\partial}{\partial x_i} \left( f(x) + \sum_{k=1}^m \lambda_k g_k(x) \right) = 0. \quad (9.11)$$

$$g_1(x) = g_2(x) = \dots = g_m(x) = 0. \quad (9.12)$$

<sup>2</sup> The number of solutions is (at least)  $\infty^1$ .

### 9.2.2 Kuhn Tucker Theorem

In 1951 an extension of the Lagrange method to cope with constraints of *inequality type* was suggested by [48]. A solution, the Kuhn Tucker theorem, to the *convex optimization* problem, i.e. to minimize a *convex* objective function under certain *convex* constraints of inequality type, was proposed.

We recall the concept of *convexness*.

**Definition 21** *The set  $A$  is called **convex** if  $\forall x, y$  it contains the interval*

$$[x, y] = \{z : z = \alpha x + (1 - \alpha)y, \quad 0 \leq \alpha \leq 1\} \quad (9.13)$$

*that connects these points.*

**Definition 22** *The function  $f$  is called **convex** if  $\forall x, y$  the inequality (**Jensen inequality**)*

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad 0 \leq \alpha \leq 1 \quad (9.14)$$

*holds true.*

We consider the following *convex optimization problem*:

**Problem 2** *Let  $X$  be a linear space, let  $A$  be a convex subset of this space, and let  $f(x)$  and  $g_k(x)$ ,  $k = 1, \dots, m$  be convex functions.*

*Minimize the function  $f(x)$  subject to the constraints*

$$x \in A \quad (9.15)$$

$$g_k(x) \leq 0 \quad k = 1, \dots, m. \quad (9.16)$$

To solve this problem we consider the *Lagrangian function*

$$\mathbb{L}(x, \lambda, \lambda_0) = \lambda_0 f(x) + \sum_{k=1}^m \lambda_k g_k(x) \quad (9.17)$$

where  $\lambda = (\lambda_1, \dots, \lambda_m)$ .

We have the following theorem.

**Theorem 16 (Kuhn Tucker)** *If  $x^*$  minimizes the function  $f(x)$  under constraints (9.15) and (9.16), then exist Lagrange multipliers  $\lambda_0^*$  and  $\lambda^* = (\lambda_1^*, \dots, \lambda_m^*)$  that are simultaneously not equal to zero and such that the following three conditions hold true:*

1. *The minimum principle*

$$\min_{x \in A} \mathbb{L}(x, \lambda_0^*, \lambda^*). \quad (9.18)$$

2. *The non-negativeness conditions*

$$\lambda_k^* \geq 0 \quad k = 0, 1, \dots, m. \quad (9.19)$$

3. *The Kuhn Tucker conditions (or Karush-Kuhn Tucker conditions)*

$$\lambda_k^* g_k(x^*) = 0, \quad k = 1, \dots, m. \quad (9.20)$$

If  $\lambda_0 \neq 0$  then conditions (1), (2) and (3) are sufficient conditions for  $x^*$  to be the solution of the optimization problem.

To get  $\lambda_0 \neq 0$ , it is sufficient that exists  $\hat{x}$  such that the following conditions (**Slater conditions**)

$$g_i(\hat{x}) < 0, \quad i = 1, \dots, m \quad (9.21)$$

holds.

This corollary follows from the Kuhn Tucker theorem.

**Corollary 2** *If the Slater conditions are satisfied, then one can choose  $\lambda_0 = 1$  and rewrite the Lagrangian in the form*

$$\mathbb{L}(x, 1, \lambda) = f(x) + \sum_{k=1}^m \lambda_k g_k(x). \quad (9.22)$$

Now the Lagrangian is defined as a function of  $n + m$  variables and conditions of the Kuhn Tucker theorem are equivalent to the existence of a saddle point  $(x^*, \lambda^*)$  of the Lagrangian, i.e.

$$\min_{x \in A} \mathbb{L}(x, 1, \lambda^*) = \mathbb{L}(x^*, 1, \lambda^*) = \max_{\lambda > 0} \mathbb{L}(x^*, 1, \lambda^*). \quad (9.23)$$

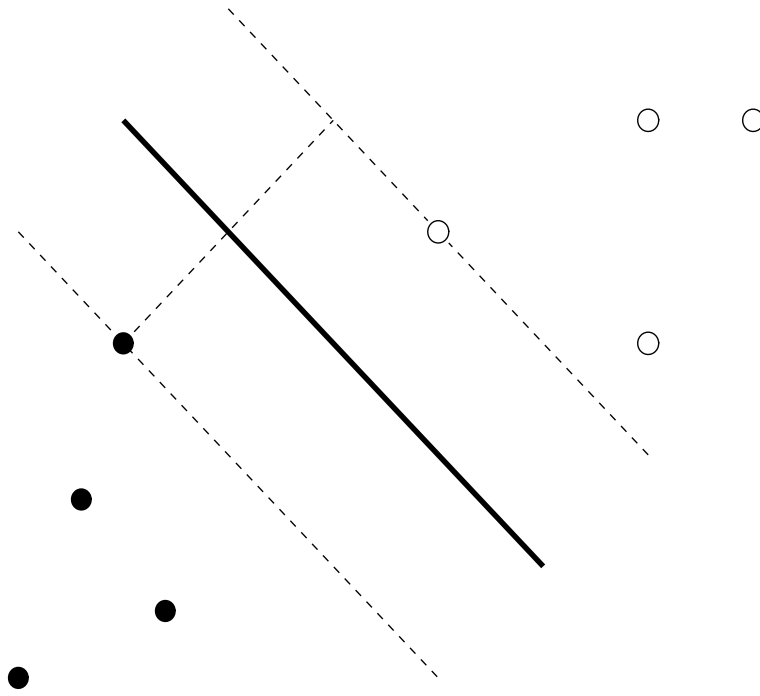
*Proof.* The left equality of (9.23) follows from conditions (1) of the Kuhn Tucker Theorem and the right equality follows from conditions (3) and (2) of the same theorem.

Lagrange Methods and Kuhn Tucker are the basic optimization tools of the kernel methods further described in the book.

### 9.3 Support Vector Machines for Classification

In this section we describe the most popular kernel method, the *support vector machines (SVM)* for classification. For the sake of the simplicity, we consider a problem of the binary classification, that is the training set has only two classes.

Let  $D$  be a training set formed by  $\ell$  patterns  $p_i$ . Each pattern  $p_i$  is a couple of values  $(\mathbf{x}_i, y_i)$  where the first term  $\mathbf{x}_i$  ( $\mathbf{x}_i \in \mathbb{R}^n$ ) is called *input* and the second term (*output*)  $y_i$  can assume only two possible discrete values, that we fix conventionally at  $+1$  and  $-1$ . The patterns with output  $+1$  are called *positive patterns*, while the others are called *negative patterns*. Finally,



**Fig. 9.2.** A binary classification problem: to separate circles from disks. The *optimal hyperplane* is orthogonal to the shortest line connecting the convex hulls of the two classes and intersects it halfway between the two classes.

we assume that each pattern  $p_i$  has been generated according to a unknown probability distribution  $P(\mathbf{x}, y)$ .

The problem of learning how to classify the patterns correctly consists in estimating a function  $f : \mathbb{R}^n \rightarrow \pm 1$  using training set patterns

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell) \in \mathbb{R}^n \times \pm 1 \quad (9.24)$$

such that  $f$  will correctly classify unseen examples  $(\mathbf{x}, y)$ , i.e.  $f(\mathbf{x}) = y$  for examples  $(\mathbf{x}, y)$  generated from the same probability distribution  $P(\mathbf{x}, y)$  of the training set. The patterns  $(\mathbf{x}_i, y_i)$  are usually assumed to be *i.i.d* i.e. *identically independent distributed*.

The underlying idea of SVM is the *optimal hyperplane algorithm*.

### 9.3.1 Optimal Hyperplane Algorithm

The class of hyperplanes

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad \mathbf{w}, \mathbf{x} \in \mathbb{R}^n, \quad b \in \mathbb{R} \quad (9.25)$$

corresponding to decision functions<sup>3</sup>

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \quad (9.26)$$

was widely discussed by [88][87]. They proposed a learning algorithm, the *generalized portrait* for linearly separable problems, that computed  $f$  from empirical data.

Besides, they observed that among all hyperplanes separating the data, there exists a unique one, the *optimal hyperplane*, yielding the maximum margin of separation between the classes

$$\max_{w,b} \min(\|\mathbf{x} - x_i\| : \mathbf{x} \in \mathbb{R}^n, \mathbf{w} \cdot \mathbf{x} + b = 0, \quad i = 1, \dots, \ell). \quad (9.27)$$

To compute the *Optimal Hyperplane* the following optimization problem has to be solved:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \quad (9.28)$$

$$\text{subject to } y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 \quad i = 1, \dots, \ell. \quad (9.29)$$

This conditional optimization problem can be solved by introducing Lagrange multipliers  $\alpha_i \geq 0$  and a Lagrangian function (see Section 9.2)  $\mathbb{L}$

$$\mathbb{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i ((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1 \quad (9.30)$$

where<sup>4</sup>  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_\ell)$ .

The Lagrangian  $\mathbb{L}$  has to be minimized with respect to the *primal variables*  $\mathbf{w}$  and  $b$  and maximized with respect to the *dual variables*  $\alpha_i$ , i.e. a saddle point has to be found. The optimization problem can be solved by means of the Kuhn Tucker theorem (see Section 9.2). The Kuhn Tucker theorem implies that the condition at the saddle point, the derivatives of  $\mathbb{L}$  with respect to the primal variables must vanish,

$$\frac{\partial \mathbb{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = 0, \quad \frac{\partial \mathbb{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \quad (9.31)$$

which leads to

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \quad (9.32)$$

and

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i. \quad (9.33)$$

<sup>3</sup> The function *signum*  $\text{sgn}(u)$  is defined as follows:  $\text{sgn}(u) = 1$  if  $u > 0$ ;  $\text{sgn}(u) = -1$  if  $u < 0$ ;  $\text{sgn}(u) = 0$  if  $u = 0$ .

<sup>4</sup> This convention is adopted in the rest of the chapter.



Hence the solution vector  $\mathbf{w}$  is an expansion in terms of a subset of the training set patterns, namely those patterns whose  $\alpha_i$  are  $\neq 0$ . These patterns are called *support vectors* (*SV*).

The Kuhn Tucker theorem implies that  $\alpha_i$  must satisfy the *Karush-Kuhn Tucker* (*KKT*) conditions

$$\alpha_i \cdot [y_i(\mathbf{x}_i \cdot \mathbf{w}_i) + b] - 1 = 0 \quad i = 1, \dots, \ell. \quad (9.34)$$

These conditions imply that the support vectors lie on the margin. All remaining samples of the training set are irrelevant for the optimization since their  $\alpha_i$  is null. This implies that the hyperplane is completely determined by the patterns closest to it, the solution should not depend on other patterns of the training set. Therefore (9.33) can be written as

$$\mathbf{w} = \sum_{\alpha_i \in SV}^{\ell} \alpha_i y_i \mathbf{x}_i. \quad (9.35)$$

Plugging (9.32) and (9.33) into  $\mathbb{L}$ , one eliminates the primal variables and the optimization problem becomes:

$$\max_{\alpha} \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (9.36)$$

$$\text{subject to} \quad \alpha_i \geq 0 \quad i = 1, \dots, \ell \quad (9.37)$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \quad (9.38)$$

Therefore the hyperplane decision function can be written as

$$f(x) = \text{sgn} \left( \sum_{i=1}^{\ell} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}_j) + b \right). \quad (9.39)$$

The optimal hyperplane algorithm can just solve linear problems. It cannot solve simple nonlinear problems as XOR, how underlined by [59]. In order to build a classifier that can solve nonlinear problems one has to find a method to perform the optimal hyperplane algorithm in a feature space nonlinearly related to the input space [1]. To this purpose, we recall the definition of *Mercer kernel* [5] (see Appendix D).

**Definition 23** *Let  $X$  be a nonempty set. A function  $G : X \times X \rightarrow \mathbb{R}$  is called a Mercer kernel (or positive definite kernel) if and only if is symmetric (i.e  $G(x, y) = G(y, x) \quad \forall x, y \in X$ ) and  $\sum_{j=1}^n \sum_{k=1}^n c_j c_k G(x_j, x_k) \geq 0$  for all  $n \geq 2$ ,  $x_1, \dots, x_n \subseteq X$  and  $c_1, \dots, c_n \subseteq \mathbb{R}$ .*

An example of the Mercer kernel is the *Gaussian*  $G(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2})$  where  $\sigma \in \mathbb{R}$ ,  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ .

The Mercer theorem (see Appendix D) states that Mercer kernels permit performing scalar products in feature spaces that are nonlinearly related to the input space. In particular, each Mercer kernel  $K(x, y)$ ,  $K : X \times X \rightarrow \mathbb{R}$  can be written as

$$K(x, y) = (\Phi(x) \cdot \Phi(y)) \quad (9.40)$$

where  $\Phi : X \rightarrow \mathcal{F}$ ,  $\mathcal{F}$  is called the *feature space*.

Hence it is adequate to substitute in the formula (9.39) the inner product  $(\mathbf{x}_i \cdot \mathbf{x}_j)$  with the Kernel  $K(\mathbf{x}_i, \mathbf{x}_j)$  to perform the optimal hyperplane algorithm in the feature space  $\mathcal{F}$ . This method is called the *kernel trick* [77].

### 9.3.2 Support Vector Machine Construction

To construct a SVM, an optimal hyperplane in some feature space has to be computed. Hence it is sufficient to substitute each training example  $\mathbf{x}_i$  with its corresponding image in the feature space  $\Phi(\mathbf{x}_i)$ . The weight vector (9.33) becomes an expansion of vectors in the feature space

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \Phi(\mathbf{x}_i). \quad (9.41)$$

Hence the weight vector is not directly computable when the mapping  $\Phi$  is unknown. Since  $\Phi(\mathbf{x}_i)$  occur only in scalar products, scalar products can be substituted by an appropriate Mercer kernel  $K$ , leading to a generalization of the decision function (9.39)

$$\begin{aligned} f(\mathbf{x}) &= \text{sgn} \left( \sum_{i=1}^{\ell} \alpha_i y_i (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) + b \right) \\ &= \text{sgn} \left( \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \end{aligned} \quad (9.42)$$

and the following quadratic problem to optimize:

$$\max_{\alpha} \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (9.43)$$

$$\text{subject to } \alpha_i \geq 0 \quad i = 1, \dots, \ell \quad (9.44)$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \quad (9.45)$$

In real-world problems due to the presence of noise, some mislabelled samples may exist and classes may be partially overlapped. Therefore it is necessary

to allow the possibility that some examples can violate (9.29). In order to get that, we introduce *slack variables* [17][85]

$$\xi_i \geq 0 \quad i = 1, \dots, \ell. \quad (9.46)$$

The slack variable  $\xi_i$  is strictly positive when the respective sample  $\mathbf{x}_i$  violates Equation (9.29); otherwise it is null. Using slack variables we can relax the constraints in the following way:

$$y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i \quad i = 1, \dots, \ell. \quad (9.47)$$

Therefore the constructed classifier, *support vector machine*, allows us to control at the same time the margin ( $\|\mathbf{w}\|$ ) and the number of training errors, given by the number of  $\xi_i \neq 0$ , by means of the minimization of the objective function:

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} \xi_i \quad (9.48)$$

subject to the constraints of (9.46) and (9.47). In Equation (9.48)  $\boldsymbol{\xi}$  stands for  $\boldsymbol{\xi} = (\xi_1, \dots, \xi_\ell)$ . The parameter  $C \geq 0$ , called *regularization constant*,<sup>5</sup> allows us to manage the trade-off between the number of the errors and the margin of hyperplane.

Plugging the constraints in (9.48) and rewriting in terms of Lagrange multipliers, we obtain the following problem to maximize

$$\max_{\boldsymbol{\alpha}} = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (9.49)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C \quad i = 1, \dots, \ell \quad (9.50)$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \quad (9.51)$$

The only difference from the separable case is the upper bound  $C$  on the Lagrange multipliers  $\alpha_i$ . As in the separable case, the decision assumes the form (9.42) The threshold  $b$  can be computed by exploiting the fact that for all SVs  $\mathbf{x}_i$  with  $\alpha_i < C$ , the slack variable  $\xi_i$  is zero, therefore

$$\sum_{i=1}^{\ell} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + b = y_i. \quad (9.52)$$

The solution of the system formed by Equations (9.49), (9.50), and (9.51) requires *quadratic programming (QP)* techniques, which are not always efficient. However, it is possible to use in SVMs different approaches that do not require QP techniques.

<sup>5</sup> The term *regularization constant* is motivated in Section 9.3.6.

### A Linear Programming Approach to Classification

Instead of using quadratic programming it is also possible to derive a kernel classifier in which the learning task involves *linear programming (LP)* instead. Whereas in the quadratic programming approach we look for the hyperplane that maximizes the margin (the optimal hyperplane), in this approach we look for the *sparsest separating hyperplane* [19] without considering the margin. An approximate solution [19] to this problem can be obtained replacing in the equation (9.48), the term  $\frac{1}{2}\|w\|^2$  with  $\sum_{i=1}^{\ell} \alpha_i$ . If we repeat the same computational strategy that we have adopted in the case of the Optimal Separating Hyperplane, after having introduced the slack variables and the kernel trick, we obtain the following linear optimization problem:

$$\min_{\alpha, \xi} \left[ \sum_{i=1}^{\ell} \alpha_i + C \sum_{i=1}^{\ell} \xi_i \right] \quad (9.53)$$

$$y_i \left[ \sum_{j=1}^{\ell} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + b \right] \geq 1 - \xi_i \quad (9.54)$$

where  $\alpha_i \geq 0$  and  $\xi_i \geq 0$  for  $i = (1, \dots, \ell)$ .

Since an efficient technique, the *simplex method* [47], is available for solving linear programming problems this approach is a practical alternative to conventional SVMs based on QP approaches. This linear programming approach [54] evolved independently of the QP approach to SVMs. It is also possible to handle multiclass problems using linear programming techniques [94].

#### 9.3.3 Algorithmic Approaches to Solve Quadratic Programming

The methods we have considered have involved linear or quadratic programming. Linear programming can be implemented using the *simplex method*. LP packages are included in the most popular mathematical software packages.

For quadratic programming there are also many applicable techniques including *conjugate gradient* and *primal-dual interior point* methods [52]. Certain QP packages are readily applicable such as MINOS and LOQO. These methods can be used to train an SVM rapidly but they have the disadvantage that the  $\ell \times \ell$  matrix  $K(\mathbf{x}_i, \mathbf{x}_j)$  (*Gram matrix*) is stored in the memory. For small datasets this is possible, but for large datasets alternatives techniques have to be used. These techniques can be grouped into three categories: techniques in which kernel components are evaluated and discarded during learning, *working set* methods in which an evolving subset of data is used, and new algorithms that explicitly exploit the structure of the problem.

For the first category the most obvious approach is to sequentially update the  $\alpha_i$  and this is the approach used by the *kernel adatron algorithm* (KA) [33].

For binary classification, with no soft margin or bias, this is a simple gradient ascent procedure on (9.49) in which  $\alpha_i \geq 0$  initially and the  $\alpha_i$  are subsequently sequentially updated using

$$\alpha_i \leftarrow \beta_i \theta(\beta_i) \quad (9.55)$$

where

$$\beta_i = \alpha_i + \eta \left[ 1 - y_i \sum_{j=1}^{\ell} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \right] \quad (9.56)$$

and  $\theta(\beta)$  is the Heaviside step function.<sup>6</sup>

The optimal learning rate  $\eta$  is

$$\frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)}. \quad (9.57)$$

A sufficient condition for the convergence is  $0 < \eta K(\mathbf{x}_i, \mathbf{x}_i) < 2$ .

Although KA is not fast as most QP routines, it is very easy to implement and it is quite useful for teaching purposes.

### Chunking and Decomposition

Rather than sequentially updating the  $\alpha_i$  the alternative is to update the  $\alpha_i$  in parallel but using only a subset or *chunk* of data at each stage. Thus a QP routine is used to optimize the Lagrangian on an initial arbitrary subset of data. The support vectors found are retained and all other datapoints (with  $\alpha_i = 0$ ) discarded. A new working set of data is then derived from these support vectors and additional datapoints which maximally violate the storage constraints. This *chunking* [63] process is then iterated until the margin is maximized. This procedure may still fail because the dataset is too large or the hypothesis modelling the data is not sparse, i.e. most  $\alpha_i$  are non-null. In this case *decomposition* [64] methods provide a better approach: these algorithms only use a fixed size subset of data with the  $\alpha_i$  for the remainder kept fixed. It is worth mentioning that SVM packages such as SVMTorch [15] and SVMLight [41] use *working set methods*.

#### 9.3.4 Sequential Minimal Optimization

The most popular decomposition method is the *sequential minimal optimization* (SMO) algorithm [66]. Several SVM packages are based on SMO or on its variants. In SMO only two  $\alpha_i$  are optimized at each iteration. If only two

<sup>6</sup>  $\theta(\beta)$  is 1 if  $\beta > 0$ , 0 otherwise.

parameters are optimized and the rest kept fixed then it is possible to derive an analytical solution which can be executed using few numerical operations. SMO is closely related to a group of optimization algorithms known as the *Bregman methods* [8] and the *row-action methods* [12][13]. We pass to describe the SMO algorithm and we fix the notation. All quantities related to the first multiplier have the subscript 1, whereas the quantities related to the second multiplier have the subscript 2. Since the multipliers are two, the multiplier constraints can be easily represented in a graphical way. The constraint implies that Lagrangian multipliers are included in a box, whereas the linear equality constraint force that Lagrange multipliers lie on a diagonal line. Hence the constrained maximum of the cost function must lie on the diagonal line. We first compute the second multiplier  $\alpha_2$  and we express the diagonal line ends in terms of  $\alpha_2$ . If  $y_1$  and  $y_2$  are not equal the constrained maximum lies on the line  $\alpha_1 - \alpha_2 = A$  hence  $\alpha_2$  must satisfy the following inequalities:

$$M = \max(0, \alpha_2 - \alpha_1); \quad N = \min(C, C + \alpha_2 - \alpha_1). \quad (9.58)$$

On the other hand, if  $y_1$  and  $y_2$  are equal, the maximum lies on the line  $\alpha_1 + \alpha_2 = A$  hence  $\alpha_2$  must satisfy the following inequalities:

$$M = \max(0, \alpha_2 + \alpha_1 - C); \quad N = \min(C, \alpha_2 + \alpha_1). \quad (9.59)$$

Now we pass to compute the constrained maximum of the cost function. If we derive the cost function (see Exercise 5) we obtain the following updating rule for the second multiplier.

$$\alpha_2(t+1) = \alpha_2(t) - \frac{y_2(E_1 - E_2)}{2K(\mathbf{x}_1, \mathbf{x}_2) - K(\mathbf{x}_1, \mathbf{x}_1) - K(\mathbf{x}_2, \mathbf{x}_2)} \quad (9.60)$$

where  $E_i = f(\mathbf{x}_i - y_i)$  and  $\alpha_2(t)$ ,  $\alpha(t)$  indicates the preceeding (*old*) and the updated value (*new*) value of the multiplier. This rule is also called *unconstrained maximum updating rule*. The *constrained maximum* can be found by limiting the unconstrained maximum to the segment ends. Thus we obtain:

$$\alpha'_2(t+1) = \left\{ \begin{array}{ll} N & \text{if } \alpha_2(t+1) \geq N \\ \alpha_2(t+1) & \text{if } M < \alpha_2(t+1) < N \\ M & \text{if } \alpha_2(t+1) \leq M. \end{array} \right\} \quad (9.61)$$

The updated value  $\alpha'_1(t+1)$  of the other multiplier can be easily obtaining remembering that the following relation, where  $s = y_1 y_2$ , has to be fulfill:

$$\alpha'_1(t+1) + s\alpha'_2(t+1) = \alpha_1(t) + s\alpha_2(t+1) \quad (9.62)$$

Therefore we obtain:

$$\alpha'_1(t+1) = \alpha_1(t) + s(\alpha_2(t) - \alpha'_2(t+1)) \quad (9.63)$$

### Strategies for Choosing Multipliers to Optimize

SMO uses heuristic strategies to pick the multipliers to optimize. SMO implements two different strategies to choose the first and the second multiplier. The choice of the first multiplier (*first choice multiplier*) represents the outer loop of the algorithm and makes the scanning of the whole training set looking for the examples which do not fulfill the KKT conditions. When such an example is found it is adopted as a candidate for optimization and it starts the search for the second multiplier. The choice of the second multiplier (*second choice multiplier*) is performed in order to maximize the step during joint optimization. In particular, SMO computes the quantity  $|E_1 - E_2|$ . If  $E_1$  is positive the example with minimum value  $E_2$  is selected. On the other hand, if  $E_1$  is negative the example with maximum value  $E_2$  is picked.

In order to make the SMO algorithm faster, the KKT conditions are relaxed. KKT conditions are fulfilled with an accuracy of  $\epsilon$  which generally assumes values such as  $10^{-2}$  or  $10^{-3}$ . Besides, further refinements of the SMO algorithm have been refined with the aim of improving its speed [45]. Finally, we conclude this section showing how the threshold  $b$  of the SVM can be computed using SMO. After each optimization step, the threshold has to be computed since the KKT conditions must be satisfied by the optimized samples. With some algebra it can obtain the following expression for the threshold  $b_1(t+1)$

$$b_1(t+1) = E_1 + y_1(\alpha_1(t+1) - \alpha_1(t))K(\mathbf{x}_1, \mathbf{x}_1) + y_2(\alpha_2'(t+1) - \alpha_2(t))K(\mathbf{x}_1, \mathbf{x}_2) + b(t) \quad (9.64)$$

which is valid when the multiplier  $\alpha_1$  is not at the bound.

Whereas the multiplier  $\alpha_2$  is not at the bound, the following expression holds:

$$b_2(t+1) = E_2 + y_1(\alpha_1(t+1) - \alpha_1(t))K(\mathbf{x}_1, \mathbf{x}_2) + y_2(\alpha_2'(t+1) - \alpha_2(t))K(\mathbf{x}_2, \mathbf{x}_2) + b(t) \quad (9.65)$$

The thresholds  $b_1(t+1)$  and  $b_2(t+1)$  are equal when they are valid. Finally, when both multipliers are at bound and if  $M$  and  $N$  are not equal, SMO select as new threshold the mean between  $b_1(t+1)$  and  $b_2(t+1)$ .

#### 9.3.5 Other Optimization Algorithms

Alternative optimization approaches have been developed. Keerthi et al. [46] have proposed a very effective binary classification algorithm based on the dual geometry of finding the two closest points in the convex hulls. These approaches have been particularly effective for linear SVM problems.

The *Lagrangian SVM (LSVM)* method of Mangasarian and Musicant [55] reformulates the classification problem as an unconstrained optimization task and then solves the problem using an algorithm which only requires the solution of systems of linear equalities. LSVM uses a method based on the *Sherman*

*Morrison Woodbury formula* which only requires solution of systems of linear equalities.

Finally it is worth mentioning the *interior-point* [27] and *semi-smooth support vector* [28] methods of Ferris and Munson that seem quite effective in solving linear classification problems with huge training sets.

### 9.3.6 SVM and Regularization Methods\*

In this section, which is addressed to an experienced reader, we discuss SVM in the framework of the *theory of regularization* [81][82]. The theory of regularization provides an effective method, the *regularization method*, to solve the so-called *ill-posed problems*. A *well-posed problem in the Hadamard sense* [38] is a problem whose solution exists, is unique and continuous<sup>7</sup>. If a problem is not well-posed, it is *ill-posed in the Hadamard sense*. In the rest of the book we adopt the convention of calling ill-posed problems in the Hadamard sense simply ill-posed problems. The problem of classification is an example of ill-posed problem. SVM for classification can be considered as a special case of regularization method. As we have seen at the beginning of this section, the problem of classification consists in estimating a function  $f : \mathbb{R}^n \rightarrow \pm 1$  using training set patterns

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell) \in \mathbb{R}^n \times \pm 1 \quad (9.66)$$

In the framework of the theory of the regularization, the problem of the classification can be represented in terms of the following minimization problem:

$$\min_{f \in \mathcal{H}} \left[ \sum_{i=1}^{\ell} L(y_i, f(\mathbf{x}_i)) + CJ(f) \right] \quad (9.67)$$

where  $L(y_i, f(\mathbf{x}_i))$  and  $J(f)$  are respectively a loss function (e.g. zero-one loss function) (see Chapter 5) and a penalty functional.  $\mathcal{H}$  is the space of functions where the penalty functional is defined.  $C \geq 0$  is called, in the theory of regularization, *regularization constant* and determines the trade-off between the loss function and the penalty term. Now, we assume that the penalty functional assumes the form

$$J(f) = \int_{\mathbb{R}^n} \frac{|F(s)|^2}{G(s)} ds, \quad (9.68)$$

where  $F$  is the Fourier transform of  $f$  and  $G$  is a positive function that  $G(s) \rightarrow 0$  as  $\|s\| \rightarrow \infty$ .

It is possible to show [37] that, using a few additional hypotheses, the solution (9.67) is

---

<sup>7</sup> In [86] the continuity requirement is replaced with the stability.



$$f(\mathbf{x}) = \sum_{j=1}^K \beta_j \psi_j(\mathbf{x}) + \sum_{i=1}^{\ell} \theta_i \hat{G}(\mathbf{x} - \mathbf{x}_i), \quad (9.69)$$

where  $\psi_j$  span the null space of the functional  $J$  and  $\hat{G}$  is the inverse Fourier transform of  $G$ . In this framework powerful statistical approximation methods such as *smoothing splines* and *thin-plate splines* [39] can be included. Another subfamily of regularization methods can be obtained by means of a Mercer kernel  $K(\mathbf{x}, \mathbf{y})$  and the associated space of function  $\mathcal{H}_K$  which is a *reproducing kernel Hilbert space (RKHS)* (see Appendix D). In this case we can express the penalty functional  $J$  of Equation (9.67). We provide a simplified description of this family of methods. The reader who is interested to this topic can refer to [24] [37] [92].

Assume that the kernel  $K(\cdot)$  can be expressed in terms of its eigenfunctions  $\psi_i$ , that is

$$K(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{\infty} \gamma_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}) \quad (9.70)$$

with  $\gamma_j \geq 0$  and  $\sum_{j=1}^{\infty} \gamma_j^2 < \infty$ .

In similar way, the elements of RKHS  $\mathcal{H}_K$  can be expressed in terms of the eigenfunctions  $\psi_j(\cdot)$ , that is:

$$f(\mathbf{x}) = \sum_{j=1}^{\infty} c_j \psi_j(\mathbf{x}) \quad (9.71)$$

with the constraint (by definition) that

$$\|f\|_{\mathcal{H}_K}^2 = \sum_{j=1}^{\infty} \frac{c_j^2}{\gamma_j} < \infty \quad (9.72)$$

where  $\|f\|_{\mathcal{H}_K}$  is defined as the norm induced by the kernel  $K$ .

In this framework the penalty functional  $J(f)$  (9.67) is assumed to be:

$$J(f) = \|f\|_{\mathcal{H}_K}^2. \quad (9.73)$$

Substituting (9.73) in (9.67) we get:

$$\min_{f \in \mathcal{H}_K} \left[ \sum_{i=1}^{\ell} L(y_i, f(\mathbf{x}_i)) + C \|f\|_{\mathcal{H}_K}^2 \right]. \quad (9.74)$$

Plugging (9.72) in (9.74) we obtain:

$$\min_{\{c_j\}_1^{\infty}} \left[ \sum_{i=1}^{\ell} L(y_i, \sum_{j=1}^{\infty} c_j \psi_j(\mathbf{x}_i)) + C \sum_{j=1}^{\infty} \frac{c_j^2}{\gamma_j} \right]. \quad (9.75)$$

It can be proven [92] that the solution of (9.74) is *finite-dimensional*, that is:

$$f(\mathbf{x}) = \sum_{j=1}^N \alpha_j K(\mathbf{x}, \mathbf{x}_j). \quad (9.76)$$

The function  $g_i(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_i)$ , viewed as function of a unique argument  $\mathbf{x}$ , is called the *representer of evaluation* at  $\mathbf{x}_i$  in  $\mathcal{H}_K$ , since for each  $f \in \mathcal{H}_K$  we have (see Appendix D):

$$\langle K(\cdot, \mathbf{x}_i), f \rangle_{\mathcal{H}_K} = f(\mathbf{x}_i). \quad (9.77)$$

In analogous way, using the reproducing property of  $\mathcal{H}_K$ , the penalty functional  $J(f)$  becomes:

$$J(f) = \sum_{i=1}^N \sum_{j=1}^N K(\mathbf{x}_i, \mathbf{x}_j) \alpha_i \alpha_j. \quad (9.78)$$

Therefore the *infinite-dimensional problem* (9.74) can be transformed, using a vector notation, in the easier *finite-dimensional problem*:

$$\min_{\boldsymbol{\alpha}} L(\mathbf{y}, \mathbb{K}\boldsymbol{\alpha}) + C\boldsymbol{\alpha}^T \mathbb{K}\boldsymbol{\alpha} \quad (9.79)$$

where  $\mathbf{y} = (y_1, \dots, y_N)$ ,  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)$  and the  $\mathbb{K}$  is the Gram matrix whose  $ij$  element is given by  $K(x_i, x_j)$ .

Support vector machines falls in the framework above described. Finally, we remark that the capacity of transforming the *infinite-dimensional problem* in a *finite-dimensional problem* is often called, in the kernel methods literature, the *kernel property*.

## 9.4 Multiclass Support Vector Machines

Support vector machines are binary classifiers. To use SVM when the number of classes  $K$  is larger than 2, some methods have been proposed [40].

### 9.4.1 One-versus-Rest Method

The first method is the *one-versus-rest* (*o-v-r*) method [74] uses a *winner takes all strategy*. A classifier is trained for each of the  $K$  classes against all the other classes. More formally, the o-v-r method consists of training  $K$  SVM classifiers  $f_j$  by labeling all training points having  $y_i = j$  with  $+1$  and  $y_i \neq j$  with  $-1$  during the training of the  $j^{\text{th}}$  classifier. In the test stage, the final decision function  $F(\cdot)$  is given by

$$F(\mathbf{x}) = \arg \max_j f_j(\mathbf{x}). \quad (9.80)$$

The computational complexity of the o-v-r method is given by  $O(K\ell^2)$  where  $\ell$  is the cardinality of the training set.

### 9.4.2 One-versus-One Method

The second method for constructing a multiclass support vector machines is the *one-versus-one* (*o-v-o*) method and uses a *voting strategy*. The method consists in learning  $\frac{K(K-1)}{2}$  classifiers. We call  $f_{ij}$  (with  $1 \leq i < j \leq K$ ) the classifier trained only by the training samples which belong to the classes  $i$  and  $j$ , labeled respectively with  $+1$  and  $-1$ . In the learning phase all  $f_{ij}$  are trained. In test phase, for each sample  $\mathbf{x}$  the *win* frequency  $w_i$  for the class  $i$  is computed by testing  $f_{ij}$  on the sample  $\mathbf{x}$  for all  $j$ . In this way, we obtain a vector  $\mathbf{w} = (w_1, \dots, w_i, \dots, w_K)$  which expresses the win frequencies of each class. Finally, the most frequent class is chosen, that is

$$F(\mathbf{x}) = \arg \max_j w_j(\mathbf{x}). \quad (9.81)$$

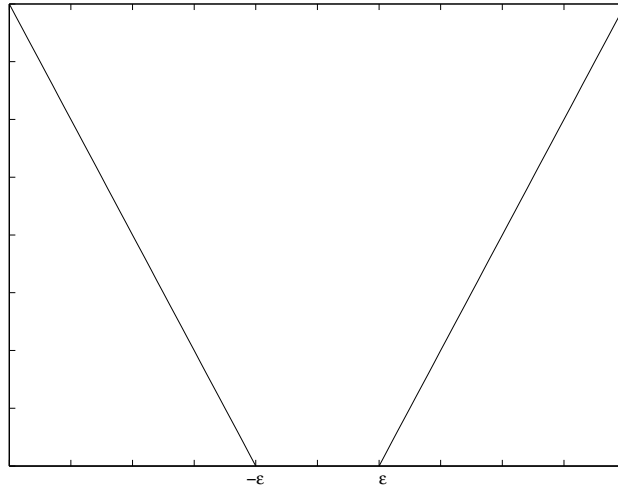
### 9.4.3 Other Methods

In addition to the o-v-o and o-v-r methods, several strategies for combining the binary SVM classifiers have been proposed. Among them we quote *DAGSVM* and the *tennis tournament method*. DAGSVM [67] consists of making a *directed acyclic graph* (*DAG*) of consecutive binary classifications. In this way a class hierarchy can be built. The final decisions are stored in the graph leaves that are obtained by exclusion. The tennis tournament method [69] produces a binary decision tree where each node is a SVM binary classifier. The decision is fixed on the basis of the rules of a tennis tournament. Therefore each class is considered a player and the winner of the match, decided on the basis of the collection of SVM pairwise classifiers trained previously, is propagated to the upper level of the tree where he will play the next match. The algorithm terminates when the root of the decision tree is reached, assigning to the unknown pattern the class which has won the last match.

## 9.5 Support Vector Machines for Regression

In this section, we extend the approach used in support vector machine for classification to the case of the regression. Whereas in the classification the output  $y$  assumes only two values ( $y \in \{\pm 1\}$ ), in the regression task the output is a real, i.e.  $y \in \mathbb{R}$ . In the regression task the underlying idea is to define a loss function that ignored errors that were within a certain distance of the true value. This type of function is referred to as an  *$\epsilon$ -insensitive loss function*.

**Definition 24** Given a data set  $\mathcal{D} = \{(\mathbf{x}_i, y_i), \dots, (\mathbf{x}_\ell, y_\ell)\} \in \mathbb{R}^n \times \mathbb{R}$  and a function  $f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ , the **linear  $\epsilon$ -insensitive loss function**  $L^\epsilon(x, y, f)$  is defined by



**Fig. 9.3.** The plot of the linear  $\epsilon$ -insensitive loss function

$$L^\epsilon(\mathbf{x}, y, f) = |y - f(\mathbf{x})|_\epsilon = \begin{cases} |y - f(\mathbf{x})| & \text{if } |y - f(\mathbf{x})| > \epsilon \\ 0 & \text{otherwise.} \end{cases} \quad (9.82)$$

where  $\mathbf{x} \in X$  and  $y \in \mathbb{R}$ . In an analogous way, the **quadratic  $\epsilon$ -insensitive loss function** is defined by

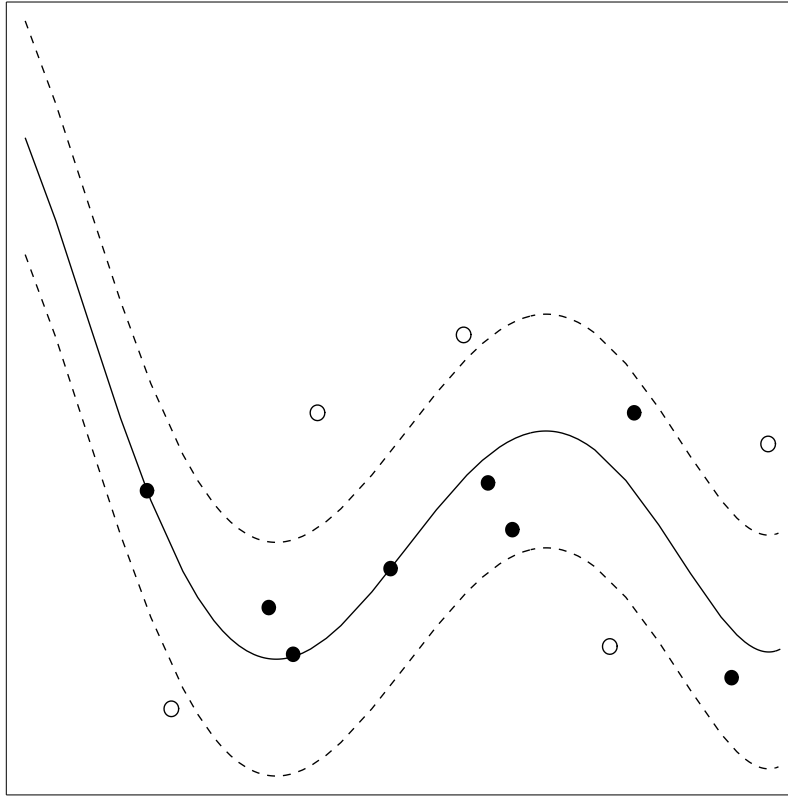
$$L^\epsilon(\mathbf{x}, y, f) = |y - f(\mathbf{x})|_\epsilon^2 = \begin{cases} |y - f(\mathbf{x})|^2 & \text{if } |y - f(\mathbf{x})|^2 > \epsilon \\ 0 & \text{otherwise.} \end{cases} \quad (9.83)$$

Figure 9.3 shows the plot of the linear  $\epsilon$ -insensitive loss function. The idea behind the  $\epsilon$ -insensitive loss function is shown in the Figure 9.4. The dotted curves delimitate a *tube* of size  $2\epsilon$  around the function  $f(x)$  and any data point outside this tube, the white circles, has a loss function not null and can be viewed as a training error. Vice versa, for the data points in the band, the black circles, the loss function is null. The above-mentioned approach is called the  $\epsilon$ -SV regression [85] and is the most common approach to SV regression, though not the only one [86].

### 9.5.1 Regression with Quadratic $\epsilon$ -Insensitive Loss

We discuss support vector machines for regression in the case of quadratic  $\epsilon$ -insensitive loss. Given a data set  $\mathcal{D} = \{(\mathbf{x}_i, y_i), \dots, (\mathbf{x}_\ell, y_\ell)\}$ , we want to estimate a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . If we assume that  $f(\cdot)$  is linear, i.e. is an hyperplane than it can be described by:

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b. \quad (9.84)$$



**Fig. 9.4.** The dotted curves delimitate a *tube* of size  $2\epsilon$  around the function  $f(x)$ . For the data points outside the tube the loss function is not null.

To solve this problem, we use the same approach of the optimal hyperplane algorithm. Therefore we minimize the following functional:

$$\tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} |y_i, f(\mathbf{x}_i)|_{\epsilon}^2 \quad (9.85)$$

where  $\mathbf{w}$  and  $C$  have the same meaning of the case of the classification task.

Comparing Equation (9.85) with (9.48) we note that we have replaced the term that expresses the number of errors in the classification with the quadratic  $\epsilon$ -insensitive loss. The regularization constant  $C$  manages the trade-off between the loss function and the the margin of the hyperplane. As in the case of the classification task, it is possible to write a constrained optimization problem defined as follows:

$$\min_{\mathbf{w}, \xi, \hat{\xi}} \left[ \|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (\xi_i^2 + \hat{\xi}_i^2) \right] \quad (9.86)$$

$$\begin{aligned}
\text{subject to} \quad & y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) \leq \epsilon + \xi_i & i = 1, \dots, \ell \\
& ((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i \leq \epsilon + \hat{\xi}_i & i = 1, \dots, \ell \\
& \xi_i \geq 0 & i = 1, \dots, \ell \\
& \hat{\xi}_i \geq 0 & i = 1, \dots, \ell \\
& \xi_i \hat{\xi}_i = 0 & i = 1, \dots, \ell, \quad (9.87)
\end{aligned}$$

where we have introduced, unlike the classification task, two slack variables  $\xi_i$  and  $\hat{\xi}_i$ . The first variable  $\xi_i$  is strictly positive when the respective pattern  $(\mathbf{x}_i, y_i)$  is such that  $f(\mathbf{x}_i) - y_i > \epsilon$ . The second variable  $\hat{\xi}_i$  is strictly positive when the respective pattern  $(\mathbf{x}_i, y_i)$  is such that  $y_i - f(\mathbf{x}_i) - y_i > \epsilon$ .

The conditional optimization problem can be solved by the usual techniques, i.e. the Lagrange Multipliers and the Kuhn Tucker Theorem, taking into account that (9.87) induces, for the corresponding Lagrange multipliers  $\alpha_i$  and  $\hat{\alpha}_i$  the relation

$$\alpha_i \hat{\alpha}_i = 0 \quad i = 1, \dots, \ell. \quad (9.88)$$

Hence we get the following objective function to maximize

$$W(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = \sum_{i=1}^{\ell} y_i (\alpha_i - \hat{\alpha}_i) - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \hat{\alpha}_i) - \frac{1}{2} \sum_{i,j=1}^{\ell} \left( (\alpha_i - \hat{\alpha}_i)(\alpha_j - \hat{\alpha}_j) ((\mathbf{x}_i \cdot \mathbf{x}_j) + \frac{1}{C} \delta_{ij}) \right) \quad (9.89)$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{i=1}^{\ell} \hat{\alpha}_i = \sum_{i=1}^{\ell} \alpha_i \\
& \alpha_i \geq 0 & i = 1, \dots, \ell \\
& \hat{\alpha}_i \geq 0 & i = 1, \dots, \ell, \quad (9.90)
\end{aligned}$$

where  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_\ell)$ ,  $\hat{\boldsymbol{\alpha}} = (\hat{\alpha}_1, \dots, \hat{\alpha}_\ell)$  and  $\delta_{ij}$  is the Kronecker symbol.<sup>8</sup>

The corresponding *KKT conditions* are

$$\begin{aligned}
\hat{\alpha}_i ((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i - \epsilon - \hat{\xi}_i &= 0 & i = 1, \dots, \ell \\
\alpha_i (y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) - \epsilon - \xi_i) &= 0 & i = 1, \dots, \ell \\
\xi_i \hat{\xi}_i &= 0 & i = 1, \dots, \ell \\
\alpha_i \hat{\alpha}_i &= 0 & i = 1, \dots, \ell. \quad (9.91)
\end{aligned}$$

If we define  $\boldsymbol{\beta} = \boldsymbol{\alpha} - \hat{\boldsymbol{\alpha}}$ , Equation (9.89) assumes a form similar to the classification case

$$\max_{\boldsymbol{\beta}} \sum_{i=1}^{\ell} y_i \beta_i - \epsilon \sum_{i=1}^{\ell} |\beta_i| - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j ((\mathbf{x}_i \cdot \mathbf{x}_j) + \frac{1}{C} \delta_{ij}) \quad (9.92)$$

<sup>8</sup>  $\delta_{ij}$  is 1 if  $i = j$ , 0 otherwise.

$$\text{subject to} \quad \sum_{i=1}^{\ell} \beta_i = 0 \quad i = 1, \dots, \ell. \quad (9.93)$$

Like in the case of the classification, we can empower the algorithm, using the kernel trick, i.e substituting in (9.92) the dot products  $(\mathbf{x}_i \cdot \mathbf{x}_j)$  with  $K(\mathbf{x}_i, \mathbf{x}_j)$  where  $K(\cdot)$  is an appropriate Mercer kernel, we get

$$\max_{\beta} \sum_{i=1}^{\ell} y_i \beta_i - \epsilon \sum_{i=1}^{\ell} |\beta_i| - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j (K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C} \delta_{ij}) \quad (9.94)$$

$$\text{subject to} \quad \sum_{i=1}^{\ell} \beta_i = 0 \quad i = 1, \dots, \ell. \quad (9.95)$$

Then the regression estimate, i.e. the function  $f(\cdot)$  modelling the data, assumes the form:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \beta_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (9.96)$$

where  $b$  can be chosen so that

$$f(\mathbf{x}_i) - y_i = -\epsilon - \frac{\beta_i}{C} \quad (9.97)$$

for any support vector  $\mathbf{x}_i$ .

### 9.5.2 Kernel Ridge Regression

We consider again the final formulation of the regression with quadratic  $\epsilon$ -insensitive loss, i.e.

$$\max_{\beta} \left( \sum_{i=1}^{\ell} y_i \beta_i - \epsilon \sum_{i=1}^{\ell} |\beta_i| - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j (K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C} \delta_{ij}) \right) \quad (9.98)$$

$$\text{subject to} \quad \sum_{i=1}^{\ell} \beta_i = 0 \quad i = 1, \dots, \ell. \quad (9.99)$$

It is necessary to make some remarks. When  $\epsilon \neq 0$ , we introduce an extra weight factor involving the dual parameters. On the other hand, when  $\epsilon$  is null the problem corresponds to considering standard least squares linear regression with a weight decay factor controlled by the regularization constant  $C$ . This approach to regression is also known as *ridge regression*, and it is equivalent to techniques derived from *Gaussian processes*, that we will examine in Section 9.6. First of all, we ignore the bias term  $b$ , since Gaussian processes do not consider the bias term. Therefore we consider the problem that can be stated as follows:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{\ell} \xi_i^2 \\ \text{subject to} \quad & y_i - (\mathbf{w} \cdot \mathbf{x}_i) = \xi_i \quad i = 1, \dots, \ell. \end{aligned} \quad (9.100)$$

Hence we derive the Lagrangian:

$$\mathbb{L}(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{\ell} \xi_i^2 + \sum_{i=1}^{\ell} \alpha_i (y_i - (\mathbf{w} \cdot \mathbf{x}_i) - \xi_i). \quad (9.101)$$

According to the optimality conditions

$$\frac{\partial \mathbb{L}}{\partial \mathbf{w}} = 0, \quad \frac{\partial \mathbb{L}}{\partial \xi_i} = 0, \quad (9.102)$$

we get

$$\mathbf{w} = \frac{1}{2\lambda} \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i \quad (9.103)$$

$$\xi_i = \frac{\alpha_i}{2}. \quad (9.104)$$

Plugging in (9.101) we have:

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_{i=1}^{\ell} y_i \alpha_i - \frac{1}{4\lambda} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \frac{1}{4} \|\boldsymbol{\alpha}\|^2 \quad (9.105)$$

and using the kernel trick, i.e. substituting  $(\mathbf{x}_i, \mathbf{x}_j)$  with the kernel  $K(\mathbf{x}_i, \mathbf{x}_j)$  where  $K(\cdot)$  is an appropriate Mercer kernel, we get the final form:

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_{i=1}^{\ell} y_i \alpha_i - \frac{1}{4\lambda} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{4} \|\boldsymbol{\alpha}\|^2. \quad (9.106)$$

Equation (9.106) can be rewritten in matricial form

$$W(\boldsymbol{\alpha}) = \mathbf{y}^T \boldsymbol{\alpha} - \frac{1}{4\lambda} \boldsymbol{\alpha}^T \mathbb{K} \boldsymbol{\alpha} - \frac{1}{4} \boldsymbol{\alpha}^T \boldsymbol{\alpha} \quad (9.107)$$

where  $\mathbf{y}$  and  $\mathbf{x}$  are the vectors formed, respectively, by  $y_i$  and  $\mathbf{x}_i$  and  $\mathbb{K}$  is the Gram matrix whose generic element  $\mathbb{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ .

If we impose

$$\frac{\partial W}{\partial \boldsymbol{\alpha}} = 0, \quad (9.108)$$

we get

$$-\frac{1}{2\lambda} \mathbb{K} \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha} + \mathbf{y} = 0. \quad (9.109)$$



Hence

$$\boldsymbol{\alpha} = 2\lambda(\mathbb{K} + \lambda\mathbb{I})^{-1}\mathbf{y} \tag{9.110}$$

where  $\mathbb{I}$  is the identity matrix.

The corresponding regression function is:

$$f(\mathbf{x}) = \mathbf{y}^T(K + \lambda\mathbb{I})^{-1}\hat{\mathbb{K}} \tag{9.111}$$

where  $\hat{\mathbb{K}}$  is the vector whose generic element is  $\mathbb{K}_i = \mathbb{K}(\mathbf{x}_i, \mathbf{x})$ .

### 9.5.3 Regression with Linear $\epsilon$ -Insensitive Loss

We discuss SVMs for regression in the case of linear  $\epsilon$ -insensitive loss. Given a data set  $\mathcal{D} = \{(\mathbf{x}_i, y_i), \dots, (\mathbf{x}_\ell, y_\ell)\}$ , we want to estimate a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . If we use the linear  $\epsilon$ -insensitive loss, we have to replace in the equation (9.85) the quadratic loss with the linear one. Therefore we have to minimize the following functional:

$$\tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} |y_i, f(\mathbf{x}_i)|_{\epsilon} \tag{9.112}$$

where  $\mathbf{w}$  and  $C$  have the same meaning of the case of the quadratic loss. As in the case of the quadratic loss, it is possible to write a constrained optimization problem defined as follows:

$$\min \left[ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (\xi_i + \hat{\xi}_i) \right] \tag{9.113}$$

$$\begin{aligned} \text{subject to} \quad & y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) \leq \epsilon + \xi_i && i = 1, \dots, \ell \\ & ((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i \leq \epsilon + \hat{\xi}_i && i = 1, \dots, \ell \\ & \xi_i \geq 0 && i = 1, \dots, \ell \\ & \hat{\xi}_i \geq 0 && i = 1, \dots, \ell \end{aligned} \tag{9.114}$$

Plugging the conditions in the equation (9.113) we get the following objective function to maximize

$$W(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = \sum_{i=1}^{\ell} y_i(\alpha_i - \hat{\alpha}_i) - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \hat{\alpha}_i) - \frac{1}{2} \sum_{i,j=1}^{\ell} (\alpha_i - \hat{\alpha}_i)(\alpha_j - \hat{\alpha}_j)(\mathbf{x}_i \cdot \mathbf{x}_j) \tag{9.115}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{i=1}^{\ell} \hat{\alpha}_i = \sum_{i=1}^{\ell} \alpha_i \\ & 0 \leq \alpha_i \leq C && i = 1, \dots, \ell \\ & 0 \leq \hat{\alpha}_i \leq C && i = 1, \dots, \ell. \end{aligned}$$

Using the kernel trick we get finally:

$$W(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = \sum_{i=1}^{\ell} y_i(\alpha_i - \hat{\alpha}_i) - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \hat{\alpha}_i) - \frac{1}{2} \sum_{i,j=1}^{\ell} (\alpha_i - \hat{\alpha}_i)(\alpha_j - \hat{\alpha}_j) K(\mathbf{x}_i, \mathbf{x}_j) \quad (9.116)$$

$$\begin{aligned} \text{subject to} \quad & \sum_{i=1}^{\ell} \hat{\alpha}_i = \sum_{i=1}^{\ell} \alpha_i \\ & 0 \leq \alpha_i \leq C \quad i = 1, \dots, \ell \\ & 0 \leq \hat{\alpha}_i \leq C \quad i = 1, \dots, \ell \end{aligned}$$

where  $K(\cdot)$  is an appropriate Mercer kernel.

Finally, we have to compute the bias  $b$ . In order to do that, we consider KKT conditions for regression. Before using the kernel trick, KKT conditions are

$$\alpha_i(\epsilon + \xi_i - y_i + (\mathbf{w} \cdot \mathbf{x}_i) + b) = 0 \quad (9.117)$$

$$\hat{\alpha}_i(\epsilon + \hat{\xi}_i + y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b) = 0 \quad (9.118)$$

where

$$\sum_{j=1}^{\ell} y_j(\alpha_j - \hat{\alpha}_j) \mathbf{x}_j = \mathbf{w} \quad (9.119)$$

$$(C - \alpha_i) \xi_i = 0 \quad (9.120)$$

$$(C - \hat{\alpha}_i) \hat{\xi}_i = 0. \quad (9.121)$$

From the latter conditions we see that only when  $\alpha_i = C$  or  $\hat{\alpha}_i = C$  the slack variables are non-null. These samples of the training set correspond to points outside the  $\epsilon$ -insensitive tube. Hence from the equation (9.119) we can find the bias from a non-bound example with  $0 < \alpha_i < C$  using  $b = y_i - (\mathbf{w} \cdot \mathbf{x}_i) - \epsilon$  and similarly for  $0 < \hat{\alpha}_i < C$  we can obtain it from  $b = y_i - (\mathbf{w} \cdot \mathbf{x}_i) + \epsilon$ . Though the bias  $b$  can be obtained using only one sample of the training set, it is better estimating the bias using an average over all points on the margin.

### 9.5.4 Other Approaches to Support Vector Regression

Apart from the formulations given here it is possible to define other loss functions giving rise to different dual objective functions. In addition, rather than specifying  $\epsilon$  a priori it is possible to specify an upper bound  $\nu$  ( $0 \leq \nu \leq 1$ ) on the fraction of the points lying outside the band and then find  $\epsilon$  by optimizing over the primal objective function

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \left( \nu \epsilon + \sum_{i=1}^{\ell} |y_i - f(\mathbf{x}_i)| \right) \quad (9.122)$$

with  $\epsilon$  acting as an additional parameter to minimize over [75].

As for classification it is possible to formulate a linear programming approach to regression with [93]

$$\min_{\alpha, \hat{\alpha}, \xi, \hat{\xi}} \left[ \sum_{i=1}^{\ell} \alpha_i + \sum_{i=1}^{\ell} \hat{\alpha}_i + \sum_{i=1}^{\ell} \xi_i + \sum_{i=1}^{\ell} \hat{\xi}_i \right] \quad (9.123)$$

subject to

$$y_i - \epsilon - \xi_i \leq \left[ \sum_{j=1}^{\ell} (\alpha_j - \hat{\alpha}_j) K(\mathbf{x}_i, \mathbf{x}_j) \right] + b \leq y_i + \epsilon + \hat{\xi}_i. \quad (9.124)$$

Minimizing the sum of the  $\alpha_i$  approximately minimizes the number of support vectors which favours sparse hypotheses with smooth functional approximations of the data. This approach does not require that  $K(\cdot)$  is a Mercer kernel [93].

## 9.6 Gaussian Processes

Gaussian processes [71] are an emerging branch of kernel methods. Unlike SVMs, that are designed to solve mainly classification problems, Gaussian processes are designed to solve essentially regression problems. Although there are some attempts [95] of using Gaussian processes for classification, the problem of solving a classification task with Gaussian processes, remains still opened.

Gaussian processes are not a novelty. In [56] a framework for regression using optimal linear estimators, within the geostatistics field, was proposed. The framework, called *kriging* in honour of a South African mining engineer, is identical to Gaussian processes, currently used in machine learning. Kriging [18] has been developed considerably in the last thirty years in geostatistics, even the been model has been developed mainly on the solution of low-dimensional problems, at most problems in  $\mathbb{R}^3$ .

Machine learning community ignored completely Gaussian processes until found them out again. it was argued , that is no reason to believe that, for real problems, neural networks should be limited to nets containing only a *small* number of hidden nodes. A neural network model with a huge number of nodes, cannot be trained with a backpropagation algorithm, based on *maximum likelihood algorithm* [23][34] (see Chapter 5), since the trained neural net *overfits* the data.

In [61] the net behavior when the number of hidden nodes goes to infinity was investigated, and was showed that it can get good performances using the *Bayesian learning* [53], instead of maximum likelihood strategy.

In the Bayesian approach to neural networks a prior distribution over the weights induces a prior distribution over functions. This prior is combined

with a noise model, which specifies the probability of observing the targets  $t_i$  given function values  $y_i$ , to yield a posterior over functions which can then be used for predictions.

In [61] it was proven that the *multilayer perceptron* [6] (see Chapter 8), will converge to a Gaussian process prior when its number of hidden nodes goes to the infinity. Although infinite networks are a method of creating Gaussian process, it is also possible to specify them directly using parametric forms for the mean and covariance functions. The advantage of the Gaussian process formulation, in comparison with infinite networks, is that the integrations, which have to be approximated for neural nets, can be carried out exactly, using matrix computations. In the following section it is described how can make regression by means of Gaussian processes.

### 9.6.1 Regression with Gaussian Processes

A stochastic process is a collection of random variables  $\{Y(\mathbf{x})|\mathbf{x} \in X\}$  indexed by a set  $X \subset \mathbb{R}^n$ . The stochastic process is specified by giving the probability distribution for every finite subsets of variables  $Y(\mathbf{x}_1), \dots, Y(\mathbf{x}_k)$  in a consistent manner.

A Gaussian process is a stochastic process which can be fully specified by its *mean function*  $\mu(\mathbf{x}) = \mathcal{E}[Y(\mathbf{x})]$  and its *covariance function*  $C(\mathbf{x}, \mathbf{x}') = \mathcal{E}[(Y(\mathbf{x}) - \mu(\mathbf{x}))(Y(\mathbf{x}') - \mu(\mathbf{x}'))]$ ; it will have a joint multivariate gaussian distribution.

In this section we consider Gaussian processes which have  $\mu(\mathbf{x}) \equiv 0$ . This is the case for many neural networks priors [61]. Otherwise it assumes that any known offset has been removed.

Given a prior covariance function  $C_P(\mathbf{x}, \mathbf{x}')$ , which can be defined by any Mercer Kernel [74], a noise process  $C_N(\mathbf{x}, \mathbf{x}')$  (with  $C_N(\mathbf{x}, \mathbf{x}') = 0$  for  $\mathbf{x} \neq \mathbf{x}'$ ) and a data set  $\mathcal{D} = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell))$ , if  $\mathbf{x} \notin \mathcal{D}$  is a test point then the respective distribution  $Y(\mathbf{x})$  has mean  $\hat{Y}(\mathbf{x})$  and variance  $\sigma_Y^2(\mathbf{x})$  given by:

$$\hat{Y}(\mathbf{x}) = \mathbf{y}^T (K_P + K_N)^{-1} k_P(\mathbf{x}) \quad (9.125)$$

$$\sigma_Y^2(\mathbf{x}) = C_P(\mathbf{x}, \mathbf{x}') + C_N(\mathbf{x}, \mathbf{x}') - k_P^T(\mathbf{x})(K_P + K_N)^{-1} k_P(\mathbf{x}) \quad (9.126)$$

where :

$$[K_P]_{ij} = C_P(\mathbf{x}_i, \mathbf{x}_j); [K_N]_{ij} = C_N(\mathbf{x}_i, \mathbf{x}_j);$$

$$k_P(\mathbf{x}) = (C_P(\mathbf{x}, \mathbf{x}_1), \dots, C_P(\mathbf{x}, \mathbf{x}_\ell))^T; \mathbf{y} = (y_1, \dots, y_n).$$

The variance  $\sigma_Y^2(x)$  provides a measure of the error that the prediction yields. If we assume that the variance of the noise process  $\sigma^2$  does not depend by the sample  $\mathbf{x}$ , we have  $K_N = \sigma^2 \mathbb{I}$ . Substituting in the previous equations we have:

$$\hat{Y}(\mathbf{x}) = \mathbf{y} (K_P + \sigma^2 \mathbb{I})^{-1} k_P(\mathbf{x}) \quad (9.127)$$

$$\sigma_Y^2(\mathbf{x}) = C_P(\mathbf{x}, \mathbf{x}') + C_N(\mathbf{x}, \mathbf{x}') - k_P^T(\mathbf{x})(K_P + \sigma^2 \mathbb{I})^{-1} k_P(\mathbf{x}). \quad (9.128)$$

The prediction value in (9.127) is the same that it is possible to obtain with a Kernel Ridge Regression, see equation (9.111), using the quadratic  $\epsilon$ -insensitive loss function. The big difference between Gaussian Processes (GP) and SVM for Regression is that GP permit computing, unlike SVM, the variance of the prediction value  $\sigma_Y^2(x)$  providing an estimate on the prediction reliability. This peculiarity makes GP very appealing for applications that require that a measure of reliability of the prediction values. Examples of these applications can be found in finance (e.g. portfolio management) and geostatistics.

## 9.7 Kernel Fisher Discriminant

In this section we describe *kernel Fisher discriminant*, namely the generalization, in the feature space, of the *Fisher discriminant* [31].

The Fisher discriminant, also called *linear discriminant analysis (LDA)*, is a classical feature extraction method (see Chapter 11) and aims to achieve an optimal linear dimensionality reduction. LDA is widely used in face recognition (see Chapter 13). We pass to describe the algorithm.

### 9.7.1 Fisher's Linear Discriminant

Let  $X_1 = (\mathbf{x}_1^1, \dots, \mathbf{x}_{\ell_1}^1)$  and  $X_2 = (\mathbf{x}_1^2, \dots, \mathbf{x}_{\ell_2}^2)$  be samples from two different classes and  $X = X_1 \cup X_2 = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  their union. We define the mean of the two classes  $\mathbf{m}_1$  and  $\mathbf{m}_2$ :

$$\mathbf{m}_1 = \frac{1}{\ell_1} \sum_{j=1}^{\ell_1} \mathbf{x}_j^1, \quad \mathbf{m}_2 = \frac{1}{\ell_2} \sum_{j=1}^{\ell_2} \mathbf{x}_j^2. \quad (9.129)$$

Fisher's linear discriminant is given by the vector  $\mathbf{w}$  which maximizes

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \quad (9.130)$$

where

$$S_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \quad (9.131)$$

$$S_W = \sum_{\mathbf{x} \in X_1} (\mathbf{x} - \mathbf{m}_1)(\mathbf{x} - \mathbf{m}_1)^T + \sum_{\mathbf{x} \in X_2} (\mathbf{x} - \mathbf{m}_2)(\mathbf{x} - \mathbf{m}_2)^T \quad (9.132)$$

$S_B$  and  $S_W$  are called the *between* and *within class scatter matrices*, respectively.

The intuition behind maximizing  $J(\mathbf{w})$  is to find a direction that maximizes the projected class means (the numerator) while minimizing the class variance in this direction (the denominator).

If we set

$$\frac{\partial J}{\partial \mathbf{w}} = 0 \quad (9.133)$$

we have:

$$(\mathbf{w}^T S_B \mathbf{w}) S_w \mathbf{w} = (\mathbf{w}^T S_W \mathbf{w}) S_B \mathbf{w} \quad (9.134)$$

From (9.131) we see that  $S_b \mathbf{w}$  is always in the direction of  $(\mathbf{m}_2 - \mathbf{m}_1)$ . We do not care about the magnitude of  $\mathbf{w}$ , only its direction. Thus we can drop any scalar factors in (9.134), we have:

$$S_w \mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1). \quad (9.135)$$

Multiplying both sides of (9.135) by  $S_w^{-1}$  we then obtain

$$\mathbf{w} \propto S_w^{-1} (\mathbf{m}_2 - \mathbf{m}_1). \quad (9.136)$$

This is known as *Fisher's linear discriminant* or *linear discriminant analysis* (LDA). Despite its name, LDA is not a discriminant but provides a direction for projection of the data onto one dimension. For this reason LDA is used as a feature extraction method, and generally represents an alternative method to the PCA (see Section 11). Nevertheless, LDA can be used to implement a linear discriminant. Indeed, the projected data  $y(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$  can subsequently be used to construct a discriminant, by choosing a threshold  $\tau$  so that we classify a new point as belonging to  $X_1$  if  $y(\mathbf{x}) \geq \tau$  and classify it as belonging to  $X_2$  otherwise. It can be proved that the vector  $w$  maximizing (9.130) has the same direction as the discriminant in the corresponding Bayes optimal classifier (see Chapter 5). Finally, for the sake of completeness, we underline that LDA can be extended to the case where there are more than two classes. In this case, the algorithm is called *multiclass LDA* [23].

### 9.7.2 Fisher Discriminant in Feature Space

Fisher discriminant is a linear algorithm. Therefore it is not effective when the data distribution is not linear. Fisher discriminant can be empowered using the same approach used for the optimal hyperplane algorithm in SVM. First we map the data nonlinearly into some Feature space  $\mathcal{F}$ , by means of an appropriate Mercer kernel, and then we compute a Fisher's linear discriminant in the feature space. In this way, we implicitly perform a nonlinear discriminant in input space.

Let  $\Phi$  be a nonlinear mapping from the input space to some feature space  $\mathcal{F}$ . To find the linear discriminant in  $\mathcal{F}$  we need to maximize

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B^\Phi \mathbf{w}}{\mathbf{w}^T S_W^\Phi \mathbf{w}} \quad (9.137)$$

where  $\mathbf{w} \in \mathcal{F}$ ,  $S_B^\Phi$  and  $S_W^\Phi$  are the corresponding matrices in  $\mathcal{F}$ :

$$S_B^\Phi = (\mathbf{m}_1^\Phi - \mathbf{m}_2^\Phi) \cdot (\mathbf{m}_1^\Phi - \mathbf{m}_2^\Phi)^T \quad (9.138)$$

$$S_W^\Phi = \sum_{\mathbf{x} \in X_1} (\Phi(\mathbf{x}) - \mathbf{m}_1^\Phi) \cdot (\Phi(\mathbf{x}) - \mathbf{m}_1^\Phi)^T + \sum_{\mathbf{x} \in X_2} (\Phi(\mathbf{x}) - \mathbf{m}_2^\Phi) \cdot (\Phi(\mathbf{x}) - \mathbf{m}_2^\Phi)^T$$

with

$$\mathbf{m}_1^\Phi = \frac{1}{\ell_1} \sum_{j=1}^{\ell_1} \Phi(\mathbf{x}_j^1), \quad \mathbf{m}_2^\Phi = \frac{1}{\ell_2} \sum_{j=1}^{\ell_2} \Phi(\mathbf{x}_j^2). \quad (9.139)$$

Since the mapping  $\Phi$  can be unknown, it is impossible to solve directly the problem. In order to overcome this difficulty we use the kernel trick, which has been successfully used in the SVMs. Instead of mapping the data explicitly we seek a formulation of the algorithm which uses only scalar products  $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$  of the training patterns which we then replace by an appropriate Mercer kernel  $K(\mathbf{x}, \mathbf{y})$ .

The theory of RKHS (see Appendix D) states that any solution  $\mathbf{w} \in \mathcal{F}$  must lie in the span of all training samples in  $\mathcal{F}$ . Therefore we can find an expansion for  $\mathbf{w}$  of the form

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i). \quad (9.140)$$

Using the expansion (9.140) and the definition of  $\mathbf{m}_1^\Phi$  and  $\mathbf{m}_2^\Phi$  we write

$$\begin{aligned} \mathbf{w}^T \mathbf{m}_i^\Phi &= \frac{1}{\ell_i} \sum_{j=1}^{\ell} \sum_{k=1}^{\ell_i} \alpha_j K(\mathbf{x}_j, \mathbf{x}_k^i) & i = 1, 2 \\ &= \boldsymbol{\alpha}^T \mathbf{M}_i & i = 1, 2 \end{aligned} \quad (9.141)$$

where we have defined

$$(M_i)_j = \frac{1}{\ell_i} \sum_{k=1}^{\ell_i} K(\mathbf{x}_j, \mathbf{x}_k^i) \quad i = 1, 2 \quad (9.142)$$

and replaced the scalar product by means of the Mercer kernel  $K(\cdot)$ .

Now we consider the numerator of (9.137). Using (9.138) and (9.141) the numerator can be rewritten as

$$\mathbf{w}^T S_B^\Phi \mathbf{w} = \boldsymbol{\alpha}^T M \boldsymbol{\alpha} \quad (9.143)$$

where

$$M = (\mathbf{M}_1 - \mathbf{M}_2)(\mathbf{M}_1 - \mathbf{M}_2)^T \quad (9.144)$$

We pass to consider the denominator. Using (9.140), the definition of  $\mathbf{m}_i^\Phi$  and a similar transformation as in (9.143), we find:

$$\mathbf{w}^T S_W^\Phi \mathbf{w} = \boldsymbol{\alpha}^T N \boldsymbol{\alpha} \quad (9.145)$$

where we set

$$N = \sum_{j=1}^2 P_j (\mathbb{I} - \mathbf{1}_{\ell_j}) P_j^T \quad (9.146)$$

$P_j$  is a  $\ell \times \ell_j$  matrix with  $(P_j)_{nm} = K(\mathbf{x}_n, \mathbf{x}_m^j)$ ,  $\mathbb{I}$  is the identity matrix and  $\mathbf{1}_{\ell_j}$  is a matrix with all elements  $\frac{1}{\ell_j}$ .

Finally combining (9.143) and (9.145), we can find Fisher's linear discriminant in the feature space  $\mathcal{F}$  by maximizing

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^T M \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T N \boldsymbol{\alpha}} \quad (9.147)$$

This problem can be solved by finding the leading eigenvector of  $N^{-1}M$ . This approach is called *kernel Fisher discriminant (KFD)* [58].

The projection of a new pattern  $\mathbf{x}$  onto  $\mathbf{w}$  is given by

$$(\mathbf{w} \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}). \quad (9.148)$$

Obviously, the proposed setting is ill-posed (see Section 9.3.6). We are estimating  $\ell$  dimensional covariance structures from  $\ell$  samples. Besides, numerical problems which cause the matrix  $N$  not to be positive, we need a way of capacity control in  $\mathcal{F}$ . In order to get that, we simply add a multiple of the identity matrix to  $N$ , i.e. replace  $N$  by  $N_\mu$  where

$$N_\mu = N + \mu \mathbb{I} \quad (9.149)$$

therefore the problem becomes to find the leading eigenvalue of  $(N_\mu)^{-1}M$ .

The use of  $N_\mu$  brings some advantages: the problem becomes numerically more stable, since for  $\mu$  large enough  $N_\mu$  become positive definite;  $N_\mu$  it can be seen in analogy to [32], decreasing the bias in sample based estimation of eigenvalues; a regularization on  $\|\boldsymbol{\alpha}\|^2$  is imposed, favoring solutions with small expansion coefficients.

## 9.8 Kernel PCA

In this section we describe *kernel principal component analysis (KPCA)*, namely the generalization, in the feature space, of the *principal component analysis (PCA)*. PCA, discussed in detail in Chapter 11.4, is a data dimensionality reduction algorithm that projects the data along the directions of maximal variance. Kernel PCA uses the same approach of SVM and kernel Fisher discriminant. First it projects data in a feature space, by means an appropriate Mercer kernel. Then it performs in the feature space the PCA algorithm. We pass to describe kernel PCA in detail.

Let  $X = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  be a data set of points in  $\mathbb{R}^n$ , KPCA algorithm consists of the following steps:



1. The *Gram* matrix  $G$  is created.  $G$  is a square matrix of rank  $\ell$ , whose generic element is  $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$  where  $\mathbf{x}_i, \mathbf{x}_j \in X$  and  $K$  is an appropriate *Mercer kernel*.
2. The matrix  $\hat{G} = (\mathbb{I} - 1_\ell)G(\mathbb{I} - 1_\ell)$  is computed. Where  $\mathbb{I}$  is the identity matrix of rank  $\ell$  and  $1_\ell$  is a square matrix of rank  $\ell$  whose elements are equal to  $\frac{1}{\ell}$ .
3. Eigenvalues and eigenvectors of matrix  $\hat{G}$  are computed.

The meaning of each step of KPCA is the following.

The first step of KPCA maps implicitly the data into a *feature space*  $\mathcal{F}$  by means of a nonlinear mapping  $\Phi$ ; second step is performed in order to assure that the data projections have zero mean; last step projects the data along the directions of maximal variance in the feature space  $\mathcal{F}$ .

### 9.8.1 Centering in Feature Space

In this subsection we show that the computation of  $\hat{G}$  assures that the data projections in feature space have zero mean, i.e.

$$\sum_{i=1}^{\ell} \Phi(\mathbf{x}_i) = 0 \quad (9.150)$$

In order to show that, we note that for any mapping  $\Phi$  and for any data set  $X = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$ , the points

$$\hat{\Phi}(\mathbf{x}_i) = \Phi(\mathbf{x}_i) - \frac{1}{\ell} \sum_{i=1}^{\ell} \Phi(\mathbf{x}_i) \quad (9.151)$$

will have zero mean in the feature space.

Hence we go on defining covariance matrix and dot product matrix  $\hat{K} = \hat{\Phi}(\mathbf{x}_i)^T \hat{\Phi}(\mathbf{x}_j)$  in the feature space  $\mathcal{F}$ .

We arrive at the eigenvalue problem

$$\hat{\lambda} \hat{\alpha} = \hat{K} \hat{\alpha} \quad (9.152)$$

with  $\hat{\alpha}$  that is the expansion coefficients of an eigenvector in the feature space  $\mathcal{F}$ , in terms of the points  $\hat{\Phi}(\mathbf{x}_i)$ , i.e.

$$\hat{V} = \sum_{i=1}^{\ell} \hat{\alpha}_i \hat{\Phi}(\mathbf{x}_i). \quad (9.153)$$

Since  $\hat{\Phi}$  can be unknown, we cannot compute  $\hat{K}$  directly; however, we can express it in terms of its noncentered counterpart  $K$ .

We consider  $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$  and we make use of the notation  $1_{ij} = 1$  for all  $i, j$ . We have:

$$\begin{aligned}
\hat{K}_{ij} &= \hat{\Phi}(\mathbf{x}_i)^T \hat{\Phi}(\mathbf{x}_j) \\
&= (\Phi(\mathbf{x}_i) - \frac{1}{\ell} \sum_{m=1}^{\ell} \Phi(\mathbf{x}_m))^T (\Phi(\mathbf{x}_i) - \frac{1}{\ell} \sum_{m=1}^{\ell} \Phi(\mathbf{x}_m)) \\
&= \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) - \frac{1}{\ell} \sum_{m=1}^{\ell} \Phi(\mathbf{x}_m)^T \Phi(\mathbf{x}_j) - \frac{1}{\ell} \sum_{n=1}^{\ell} \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_n) + \frac{1}{\ell^2} \sum_{m,n=1}^{\ell} \Phi(\mathbf{x}_m)^T \Phi(\mathbf{x}_n) \\
&= G_{ij} - \frac{1}{\ell} \sum_{n=1}^{\ell} 1_{im} G_{mj} - \frac{1}{\ell} \sum_{n=1}^{\ell} 1_{nj} G_{mj} + \frac{1}{\ell^2} \sum_{n,m=1}^{\ell} 1_{im} G_{mn} 1_{nj} \tag{9.154}
\end{aligned}$$

If we define the matrix  $(1_{\ell})_{ij} = \frac{1}{\ell}$  and  $\mathbb{I}$  the Identity matrix, we have:

$$\begin{aligned}
\hat{K}_{ij} &= G - 1_{\ell} G - G 1_{\ell} + 1_{\ell} G 1_{\ell} \\
&= \mathbb{I} G - 1_{\ell} G + (1_{\ell} G - G) 1_{\ell} \\
&= (\mathbb{I} - 1_{\ell}) G + (1_{\ell} G - \mathbb{I} G) 1_{\ell} \\
&= (\mathbb{I} - 1_{\ell}) G \mathbb{I} - (\mathbb{I} - 1_{\ell}) G 1_{\ell} \\
&= (\mathbb{I} - 1_{\ell}) G (\mathbb{I} - 1_{\ell}) \\
&= \hat{G} \tag{9.155}
\end{aligned}$$

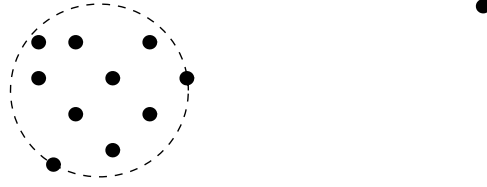
An immediate result, since the projections of data are zero mean, is the following:

**Remark 1** *The matrix  $\hat{G}$  is singular.*

*Proof.* The elements of the matrix  $C = \mathbb{I} - 1_{\ell}$  are equal to  $1 - \frac{1}{\ell}$  if they are on the diagonal, Otherwise they are equal to  $-\frac{1}{\ell}$ . If we sum the rows of  $C$  we get the null row. Therefore the determinant of  $C$  is null since its rows are linearly dependent. The determinant of  $\hat{G}$  is also null, for *Binet* [47] theorem. Hence  $\hat{G}$  is singular and has at least *one* null eigenvalue.

The remark implies that *at least* the last eigenvector, i.e the eigenvector associated to the smallest eigenvalue, must be discarded. Besides, the remark provides a requirement, that is the smallest eigenvalue of  $\hat{G}$  is null, that the eigenvalue spectrum should satisfy. The computation of eigenvalues and eigenvector of  $\hat{G}$  requires the matrix diagonalization, that can be computationally cumbersome when the rank of  $\hat{G}$  is high.

In [73] a computationally efficient method, based on the EM algorithm [20], has been proposed for extract eigenvalues and eigenvectors. The algorithm seems to overcome the above mentioned bottleneck. Finally, if KPCA is performed with the Gaussian kernel (*GKPCA*), a theoretical result has been established. In [84] It has been proven that GKPCA, in the case of an infinite number of data points, approaches to PCA, for large values of the variance  $\sigma$ . Finally, we conclude the section remarkin that kernel PCA is widely used, as feature extraction method, in face recognition (see Chapter 13).



**Fig. 9.5.** The dotted circle encloses almost data points of the figure, i.e. all the data with the exception of the two outliers.

## 9.9 One-Class SVM

*One-class SVM* [77] [80] is a unsupervised kernel method based on support vector description of a data set. In One-class SVM there are no negative examples; therefore all data are considered positive examples. One-class SVM has been initially proposed to estimate the *support distribution function* of a data set, i.e. a function that takes positive value +1 in the region that contains most data and -1 otherwise. For this reason, One Class SVM is generally applied to solve *novelty detection* problems [3] and to detect outliers. The aim of One-class SVM is to look for the smallest sphere enclosing almost all images, in the feature space, of data points, i.e all images without the outliers (see figure 9.5). Let  $X = (\mathbf{x}_1, \dots, \mathbf{x}_\ell) \subseteq \mathbb{R}^n$  be a data set. Using a nonlinear transformation  $\Phi$  from the input space to some high-dimensional feature space  $\mathcal{F}$ , it looks for the smallest enclosing sphere of radius  $R$ . This is described by the constraints:

$$\|\Phi(\mathbf{x}_j) - \mathbf{a}\|^2 \leq R^2 \quad \forall j \quad (9.156)$$

where  $\|\cdot\|$  is the Euclidean norm and  $\mathbf{a}$  is the center of the sphere.

The constraints can be relaxed using *slack* variables  $\xi_j$ :

$$\|\Phi(\mathbf{x}_j) - \mathbf{a}\|^2 \leq R^2 + \xi_j \quad (9.157)$$

with  $\xi_j \geq 0$ .

In order to solve the problem the *Lagrangian* is introduced:

$$\mathbb{L} = R^2 - \sum_{j=1}^{\ell} (R^2 + \xi_j - \|\Phi(\mathbf{x}_j) - \mathbf{a}\|^2) \beta_j - \sum_{j=1}^{\ell} \xi_j \mu_j + C \sum_{j=1}^{\ell} \xi_j \quad (9.158)$$

where  $\beta_j \geq 0$  and  $\mu_j \geq 0$  are Lagrange multipliers,  $C$  is a constant and  $C \sum_{j=1}^{\ell} \xi_j$  is a penalty term.

If we put

$$\frac{\partial \mathbb{L}}{\partial R} = 0; \quad \frac{\partial \mathbb{L}}{\partial \mathbf{a}} = 0; \quad \frac{\partial \mathbb{L}}{\partial \xi_j} = 0 \quad (9.159)$$

we get

$$\sum_{j=1}^{\ell} \beta_j = 1 \quad (9.160)$$

$$\mathbf{a} = \sum_{j=1}^{\ell} \beta_j \Phi(\mathbf{x}_j) \quad (9.161)$$

$$\beta_j = C - \mu_j. \quad (9.162)$$

The Karush-Kuhn Tucker conditions yield

$$\xi_j \mu_j = 0 \quad (9.163)$$

$$(R^2 + \xi_j - \|\Phi(\mathbf{x}_j) - \mathbf{a}\|^2) \beta_j = 0. \quad (9.164)$$

It follows from (9.164) that the image of a point  $\mathbf{x}_j$  with  $\xi_j > 0$  and  $\beta_j > 0$  lies outside the feature space sphere. Equation (9.163) states that such a point has  $\mu_j = 0$ , hence we conclude from Equation (9.162) that  $\beta_j = C$ . This will be called a *bounded support vector (BSV)*. A point  $\mathbf{x}_j$  with  $\xi_j = 0$  is mapped to the inside or to the surface of the feature space sphere. If its  $0 < \beta_j < C$  then (9.164) implies that its image  $\Phi(\mathbf{x}_j)$  lies on the surface of the feature space sphere. Such a point will be referred to as a *support vector (SV)*. support vectors lie on cluster boundaries, BSVs lie outside the boundaries and all other points lie inside them. The constraint (9.160) implies when  $C \geq 1$  no BSVs exist. Using these relations we may eliminate the variables  $R$ ,  $\mathbf{a}$  and  $\mu_j$ , turning the Lagrangian into the Wolfe dual form that is a function of the variables  $\beta_j$ :

$$W = \sum_{j=1}^{\ell} \Phi(\mathbf{x}_j)^2 \beta_j - \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j). \quad (9.165)$$

Since the variables  $\mu_j$  do not appear in the Lagrangian they may be replaced with the constraints:

$$0 \leq \beta_j \leq C \quad j = 1, \dots, \ell. \quad (9.166)$$

We compute the dot products  $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$  by an appropriate Mercer kernel  $G(\mathbf{x}_i, \mathbf{x}_j)$ . Therefore the Lagrangian  $W$  becomes

$$W = \sum_{j=1}^{\ell} G(\mathbf{x}_j, \mathbf{x}_j) \beta_j - \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j G(\mathbf{x}_i, \mathbf{x}_j). \quad (9.167)$$

At each point  $\mathbf{x}$  the distance  $D$  of its image in the feature space from the center of the sphere is given by :

$$D^2(\mathbf{x}) = \|\Phi(\mathbf{x}) - \mathbf{a}\|^2. \quad (9.168)$$

Using (9.161) we have:

$$D^2(\mathbf{x}) = G(\mathbf{x}, \mathbf{x}) - 2 \sum_{j=1}^{\ell} \beta_j G(\mathbf{x}_j, \mathbf{x}) + \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j G(\mathbf{x}_i, \mathbf{x}_j). \quad (9.169)$$

The radius of the sphere  $R$  is just the distance between a support vector and the center  $\mathbf{a}$ .

### 9.9.1 One-Class SVM Optimization

In the previous section we have just formulated the support vector machines using a problem of quadratic programming. The problem can be solved using QP packages when the dimension of the training set is quite limited. In other cases, the best solution is to use a modified version of SMO (see Section 9.3.4) [77].

The strategy of SMO is to break up the constrained minimization of (9.167) into the smallest optimization step possible. Due to the constraint on the sum of the dual variables, it is impossible to modify individual variables separately without possibly violating the constraint. Therefore the optimization has to be performed over pairs of multipliers. The algorithm is based on an *elementary optimization step*.

#### Elementary Optimization Step

For instance, consider optimizing over  $\alpha_1$  and  $\alpha_2$  with all other variables fixed. If we define  $G_{ij} = G(\mathbf{x}_i, \mathbf{x}_j)$ , Equation (9.167) becomes:

$$\min_{\alpha_1, \alpha_2} \frac{1}{2} \sum_{i=1}^2 \sum_{j=1}^2 \alpha_i \alpha_j G_{ij} + \sum_{i=1}^2 \alpha_i C_i + C, \quad (9.170)$$

where

$$C_i = \sum_{j=3}^{\ell} \alpha_j G_{ij}, \quad C = \sum_{i=3}^{\ell} \sum_{j=3}^{\ell} \alpha_i \alpha_j G_{ij} \quad (9.171)$$

subject to

$$0 \leq \alpha_1 \leq \frac{1}{\nu \ell} \quad (9.172)$$

$$0 \leq \alpha_2 \leq \frac{1}{\nu \ell} \quad (9.173)$$

$$\sum_{i=1}^2 \alpha_i = \Delta = 1 - \sum_{i=1}^3 \alpha_i. \quad (9.174)$$

We discard  $C$ , which is independent of  $\alpha_1$  and  $\alpha_2$ , and eliminate  $\alpha_1$  to obtain

$$\min_{\alpha_2} W(\alpha_2) = \frac{1}{2}(\Delta - \alpha_2)^2 G_{11} + (\Delta - \alpha_2) \alpha_2 G_{12} + \frac{1}{2} \alpha_2^2 G_{22} + (\Delta - \alpha_2) C_1 + \alpha_2 C_2. \quad (9.175)$$

Computing the derivative of  $W$  and setting it to zero, we have:

$$-(\Delta - \alpha_2)G_{11} + (\Delta - 2\alpha_2)G_{12} + \alpha_2 G_{22} - C_1 + C_2 = 0. \quad (9.176)$$

Solving the equation for  $\alpha_2$ , we get:

$$\alpha_2 = \frac{\Delta(G_{11} - G_{12}) + C_1 - C_2}{G_{11} + G_{22} - 2G_{12}}. \quad (9.177)$$

Once  $\alpha_2$  is found,  $\alpha_1$  can be recovered from  $\alpha_1 = \Delta - \alpha_2$ . If the new point  $(\alpha_1, \alpha_2)$  is outside of  $\left[0, \frac{1}{\nu\ell}\right]$ , the constrained optimum is found by projecting  $\alpha_2$  from (9.177) into the region allowed by the constraints and recomputing  $\alpha_1$ . The offset is recomputed after every such step. Additional insight can be obtained by rewriting the last equation in terms of the outputs of the kernel expansion on the examples  $x_1$  and  $x_2$  before the optimization step.

Let  $\alpha_1^*$ ,  $\alpha_2^*$  denote the values of their Lagrange parameters before the step. Then the corresponding outputs

$$O_i = G_{1i}\alpha_1^* + G_{2i}\alpha_2^* + C_i. \quad (9.178)$$

Using the latter to eliminate the  $C_i$ , we end up with an update equation for  $\alpha_2$  which does not explicitly depend on  $\alpha_1^*$ ,

$$\alpha_2 = \alpha_2^* + \frac{O_1 - O_2}{G_{11} + G_{22} - 2G_{12}}, \quad (9.179)$$

which shows that the update is essentially the fraction of first and second derivative of the objective function along the direction of the constraint satisfaction. Clearly, the same elementary optimization step can be applied to any pair of two variables, not just  $\alpha_1$  and  $\alpha_2$ . We next briefly describe how to do the overall optimization.

### SMO Optimization Algorithm

The initialization of the algorithm is the following. We start by setting a random fraction  $\nu$  of all  $\alpha_i$  to  $\frac{1}{\nu\ell}$ . If  $\nu\ell$  is not an integer, then one of the examples is set to a value in  $\left(0, \frac{1}{\nu\ell}\right)$  to ensure that  $\sum_{i=1}^{\ell} \alpha_i = 1$ . Besides, we set the initial  $\rho$  to

$$\rho = \max_{i \in [\ell], \alpha_i > 0} O_i. \quad (9.180)$$

Then we select a first variable for the elementary optimization step in one of the two following ways. Here, we use the shorthand  $SV_{nb}$  for the indices of variables which are not at bound, i.e.

$$SV_{nb} = \left\{ i : i \in [\ell], 0 < \alpha_i < \frac{1}{\nu\ell} \right\}. \quad (9.181)$$

These correspond to points that will sit exactly on the hyperplane, that will therefore have a strong influence on its precise position. The couple of the parameters on which applying the elementary optimization algorithm is selected by using the following heuristics:

1. We scan over the entire dataset until we find a variable violating a KKT condition, i.e. a point such that

$$(O_i - \rho)\alpha_i > 0, \quad (9.182)$$

or

$$(\rho - O_i) \left( \frac{1}{\nu\ell} - \alpha_i \right) > 0. \quad (9.183)$$

Once we have found one, say  $\alpha_i$ , we pick  $\alpha_j$  according to:

$$j = \arg \max_{n \in SV_{nb}} |O_i - O_n|. \quad (9.184)$$

2. The same as the above item, but the scan is only performed over  $SV_{nb}$ .

One scan of the first type is followed by multiple scans of the second type. If the first type scan finds no KKT violations, the optimization terminates. In unusual circumstances, the choice heuristic cannot make positive progress. Therefore, a hierarchy of other choice heuristics is applied to ensure positive progress. These other heuristics are the same as in the case of classification. SMO usually converges in most cases. However to ensure convergence, even in rare pathological conditions, the algorithm can be modified slightly [45].

## 9.10 Kernel Clustering Methods

In this section we present some clustering methods based on kernels. We describe the kernel extension of K-Means, the so-called *kernel K-Means*, some extensions of one-class SVM and spectral clustering methods.

### 9.10.1 Kernel K-Means

In this section we describe how the classical algorithm K-Means (see Chapter 6) can be reformulated in the feature space. In this section we use the formalism proposed by [9]. Given a data set  $X = (\mathbf{x}_1, \dots, \mathbf{x}_\ell) \subseteq \mathbb{R}^n$ , we map

our data in some Feature Space  $\mathcal{F}$ , by means a nonlinear map  $\Phi$ . We call the set  $\mathcal{A} = (\mathbf{a}_1, \dots, \mathbf{a}_K)$  *feature space codebook* since in our representation the centers in the feature space play the same role of the codebook (see Chapter 6) in the input space. In analogy with the codevectors in the input space, we define for each center  $\mathbf{a}_c$  its *Voronoi region* and *Voronoi set* in feature space. The *Voronoi region in feature space* ( $FR_c$ ) of the center  $\mathbf{a}_c$  is the set of all vectors in  $\mathcal{F}$  for which  $\mathbf{a}_c$  is the closest vector

$$FR_c = \{\boldsymbol{\xi} \in \mathcal{F} \mid c = \arg \min_j \|\boldsymbol{\xi} - \mathbf{a}_j\|\}. \quad (9.185)$$

The *Voronoi Set in feature space* ( $FV_c$ ) of the center  $\mathbf{a}_c$  is the set of all vectors  $\mathbf{x}_i$  in  $X$  such that  $\mathbf{a}_c$  is the *closest vector* for their images  $\Phi(\mathbf{x}_i)$  in the feature space

$$FV_c = \{\mathbf{x}_i \in X \mid c = \arg \min_j \|\Phi(\mathbf{x}_i) - \mathbf{a}_j\|\} \quad (9.186)$$

These definitions induce a *Voronoi tessellation of the feature space*. It is also possible to define the *empirical quantization error in feature space* defined by:

$$J(\mathcal{A}, X) = \sum_{i=1}^K \sum_{\mathbf{x} \in FV_i} \|\Phi(\mathbf{x}) - \mathbf{a}_i\|^2 \quad (9.187)$$

We pass to describe Kernel K-Means which has the following steps:

1. Project the data set  $X$  into a feature space  $\mathcal{F}$  by means a mapping  $\Phi$ . Initialize the feature space Codebook  $\mathcal{A}$ .
2. Compute for each center  $\mathbf{a}_i$  its feature Voronoi set  $FV_i$ .
3. Update each center with the mean of its feature Voronoi set, that is

$$\mathbf{a}_i = \frac{1}{|FV_i|} \sum_{\mathbf{x} \in FV_i} \Phi(\mathbf{x}) \quad (9.188)$$

4. Go to step 2 if any  $\mathbf{a}_i$  changes otherwise return the feature space codebook.

Kernel K-Means minimizes the empirical quantization error in feature space. It is necessary to remark that even we do not know the  $\Phi$  we are always able to compute the Voronoi set in the feature space. In fact the distance between any center and any sample  $\mathbf{x}$ , using the kernel trick is given by:

$$\|\Phi(\mathbf{x}) - \mathbf{a}_i\|^2 = K(\mathbf{x}, \mathbf{x}) - 2 \sum_{r=1}^K G(\mathbf{x}, \mathbf{x}_r) - \sum_{r=1}^K \sum_{s=1}^K G(\mathbf{x}_s, \mathbf{x}_r) \quad (9.189)$$

where  $G(\cdot)$  is an appropriate Mercer kernel.

The term kernel K-Means has been used in several contexts. In [76] this term was used, for the first time, for an algorithm which we will discuss in Section 9.10.3. In [36] a different formulation for kernel K-Means has been proposed. A typical formalism of *fuzzy clustering algorithms* (See Section 6.8)



has been used, i.e.  $c$  denotes the number of the codevectors and a *membership matrix*  $U$  has been introduced. Each element  $u_{ik}$  denotes the membership of the sample  $\mathbf{x}_i$  to the feature Voronoi set  $FV_i$ . The algorithm tries to minimize the empirical quantization error in feature space which rewritten as :

$$J(\mathcal{A}, X, U) = \sum_{i=1}^c \sum_{j=1}^{\ell} u_{ij} \|\Phi(\mathbf{x}_j) - \mathbf{a}_i\|^2 \quad (9.190)$$

The minimization technique used by [36] is *deterministic annealing* [72] which is a stochastic method for optimization. The minimization algorithm provides the following membership matrix:

$$u_{ij} = \frac{\exp(-\beta \|\mathbf{x}_i - \mathbf{a}_j\|^2)}{\sum_{j=1}^c \exp(-\beta \|\mathbf{x}_i - \mathbf{a}_j\|^2)}. \quad (9.191)$$

The parameter  $\beta \in \mathbb{R}$  controls the softness of the membership during the optimization and can be thought proportional to the inverse of the temperature of a physical system. This parameter is gradually increased during the annealing and at the end of the procedure the memberships have become *crisp* (see Section 6.8) and therefore a tessellation in feature space is produced. This linear partitioning in  $\mathcal{F}$ , back to the input space, forms a nonlinear partitioning of the input space.

### 9.10.2 One-Class SVM Extensions

In this section we present two extensions of one-class SVM which have been proposed for clustering.

#### Support Vector Clustering

*Support vector clustering (SVC)* [4] is an extension of one-class SVM. SVC is composed of two steps. The first step of SVC consists in performing One Class SVM. The second step of SVC is a cluster assignment procedure, based on a geometric idea. Any path connecting a pair of points belonging to different clusters must exit from the sphere in the feature space. These paths have a segment of points  $s$  such that  $R(s) > R$ . Let  $Y$  be the path connecting two points in the feature space, the following adjacency relation can be defined:

$$Y = \left\{ \begin{array}{l} 1 \text{ if } R(s) < R \\ 0 \text{ otherwise.} \end{array} \right\} \quad (9.192)$$

Clusters are provided by the connected components of the graph whose adjacency matrix is defined by (9.192). Recently, some modifications [96][50] of the labelling procedure, which seems to improve the performances, have been proposed. Finally, an improved version of the SVC algorithm applied to the handwritten recognition has been proposed in [14].

### Camastra Verri Algorithm

Another technique that combines one-class SVM and K-Means has been proposed in [9]. This method, called for the sake of simplicity the *Camastra Verri algorithm*, considers a codebook in feature space and uses a K-Means-like strategy, that is moves the centers  $a_i$  of the codebook in the feature space computing one-class SVM on their Voronoi sets  $FV_i$  until no center changes anymore.

To make robust the algorithm with respect to the outliers one-class SVM, which we call for simplicity *1-SVM*, is computed on  $FV_c(\rho)$  of each center  $a_c$ .  $FV_c(\rho)$  is defined as

$$FV_c(\rho) = \{\mathbf{x}_i \in FV_c \text{ and } \|\Phi(\mathbf{x}_i) - \mathbf{a}_c\| < \rho\} \quad (9.193)$$

$FV_c(\rho)$  is the Voronoi set in the feature space of the center  $a_c$  without outliers, that is the images of data points whose distance from the center is larger than  $\rho$ . The parameter  $\rho$  can be set up using model selection techniques [6].

Camastra Verri algorithm has the following steps:

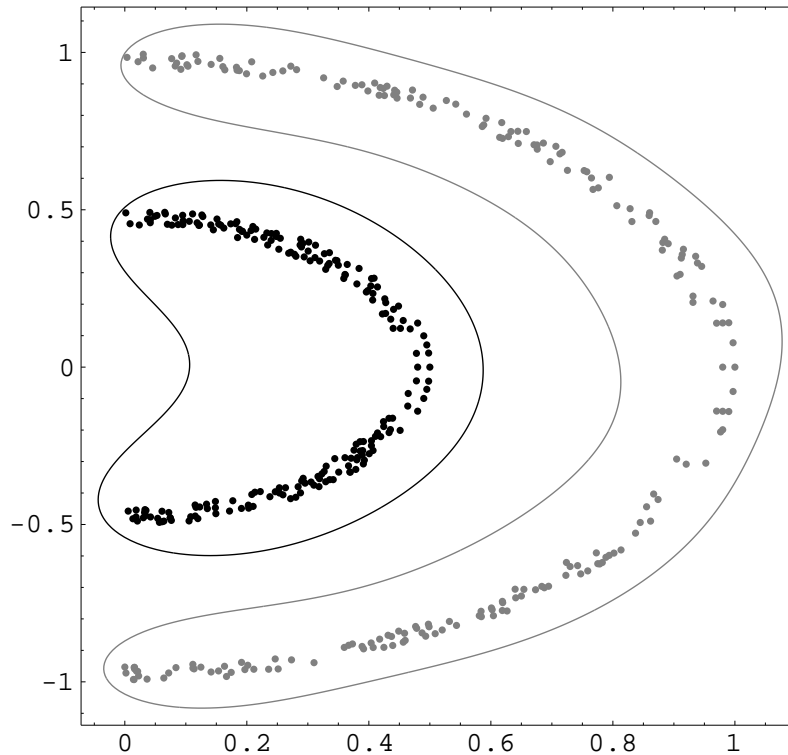
1. Project the data set  $X$  into a feature space  $\mathcal{F}$ , by means a nonlinear mapping  $\Phi$ . Initialize the centers  $\mathbf{a}_c \quad c = 1, \dots, K \quad \mathbf{a}_c \in \mathcal{F}$
2. Compute for each center  $\mathbf{a}_c$   $FV_c(\rho)$
3. Apply one-class SVM to each  $FV_c(\rho)$  and assign to  $\mathbf{a}_c$  the center yielded, i.e.  $\mathbf{a}_c = 1SVM(FV_c(\rho))$
4. Go to step 2 until any  $\mathbf{a}_c$  changes
5. Return the feature space codebook.

The second step is the expectation stage of an EM algorithm. With regard to the third step, when the constant  $C$  is taken not lower than 1, one-class SVM computes the smallest ball that encloses all data. Intuitively under this condition the third step is the maximization stage of an EM algorithm and the algorithm convergence is guaranteed, since each EM algorithm is convergent. Besides, the authors claims that the algorithm, with  $C \geq 1$  and a  $\rho$  fixed during the different iterations, has always converged in all experiments.

#### 9.10.3 Spectral Clustering

Finally, we conclude the section on kernel methods describing briefly spectral clustering methods. Although these have not been developed in the framework of the kernel methods, they have strong connections with them. It has been shown [22][21] that spectral clustering, under given conditions, is perfectly equivalent to kernel K-Means. For this reason, it is convenient that spectral clustering methods are included in the family of kernel methods for clustering.

Spectral clustering methods [2][7][19][30][43][57][62] [79] have a strong connections with the graph theory. Spectral clustering methods have widely applied into several applicative domains (e.g. image segmentation [79] and bioin-



**Fig. 9.6.** Camastra and Verri algorithm applied to a nonlinear separable data set. The black and the grey curves delimitate the two feature Voronoi sets produced by the algorithm. The data set cannot be separable, using two codevectors, by means of classical clustering algorithms such as K-Means, SOM, neural gas.

formatics [65]). Besides, the *consistency* of spectral clustering has been recently proven [90, 91] showing in this way that spectral clustering is theoretically well-grounded. Now, we pass to introduce spectral clustering algorithms describing in detail the most popular clustering algorithm, namely the *Ng-Jordan algorithm* [62].

Let  $X = (\mathbf{x}_1, \dots, \mathbf{x}_\ell) \subseteq \mathbb{R}^n$  be the data, we can build a weighted undirected graph  $\mathcal{G}$  starting from  $X$  where each sample is represented by means of a node. The distance (or *adjacency*)  $a_{ij}$  between two nodes  $\mathbf{x}_i$  and  $\mathbf{x}_j$  is defined by:

$$W_{ij} = \left\{ \begin{array}{ll} h(\mathbf{x}_i, \mathbf{x}_j) & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{array} \right\}. \quad (9.194)$$

The function  $h(\cdot)$  measures the dissimilarity between data. In this framework clustering can be viewed as a *graph cut problem* and the spectral theory permit relaxing the complexity of the problem.

For the weighted graph  $\mathcal{G}$  we call the weight matrix  $W$ , whose elements are provided by (9.194), the *adjacency matrix* (or *affinity matrix*) of  $\mathcal{G}$ . Then we define the diagonal matrix  $D$  whose generic element  $D_{ii}$  is the sum of the  $i$ -th row of the matrix  $A$ . Being said that, Ng-Jordan algorithm is formed by the following steps:

1. Choose as dissimilarity function the gaussian kernel, namely  $h(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2})$  ( $\sigma \in \mathbb{R}$ ) and build the affinity matrix  $A$ .
2. Compute the  $D$  matrix and construct the matrix  $L = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ .
3. Compute the  $k$  largest eigenvectors of  $L$   $\mathbf{e}_1, \dots, \mathbf{e}_k$  and build the matrix  $E = [\mathbf{e}_1, \dots, \mathbf{e}_k]$ .
4. Compute the matrix  $Y$  from  $E$  normalizing each of rows of  $E$  in order to have unit length, i.e the element  $ij$  of the matrix  $Y$  is given by:

$$Y_{ij} = \frac{E_{ij}}{\sum_{j=1}^k E_{ij}^2} \quad (9.195)$$

5. Defining a new data set  $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_\ell\}$ , belonging to  $\mathbb{R}^k$ , which are provided by the  $Y$  rows, namely the  $i$ -point  $\mathbf{p}_i$  is given by the  $i$ -th row of  $Y$ . Cluster  $\mathcal{P}$  into  $k$  clusters using a clustering algorithm (e.g. K-Means).
6. Assign the original point  $\mathbf{x}_i$  to the cluster  $j$  iff the point  $\mathbf{p}_i$  was assigned to the cluster  $j$ .

The Ng-Jordan algorithm has a strong analogy to the idea proposed, but not fully investigated, by [76] in their early technical report about the Kernel PCA. They have proposed an algorithm (*kernel K-Means*) which consists in applying the kernel PCA on the data and then clustering the projected data along the largest kernel eigenvectors by means of K-Means.

Finally, we recall that other spectral clustering approaches have been proposed (see [29] for a review). In particular, we quote the *Meila and Shi algorithm* based on the framework of *Markov random walks* [57] and the *Shi and Malik algorithm* [79], based on the optimal partitioning of the graph by means of the minimization of the graph cut.

## 9.11 Software Packages

We conclude the chapter providing a brief survey of the public domain software packages which implement kernel methods. The most popular packages are *SVM<sup>Light</sup>*, *SVM<sup>Torch</sup>* and *LIBSVM*.

*SVM<sup>Light</sup>* [41] can be downloaded from [svmlight.joachims.org](http://svmlight.joachims.org). It implements support vector for classification and for regression. It is also available a SVM version (*SVM<sup>struct</sup>*)[83] for multivariate and structured outputs like trees and sequences.

SVMTorch was developed by [15]. At present, it is integrated in the machine learning library *Torch* [16], which can be downloaded at [www.torch.ch](http://www.torch.ch). SVMTorch, written in C++, implements support vector for classification and for regression.

LIBSVM [25], written in C++, is a public domain library for support vector machines and is downloadable from [www.csie.ntu.edu.tw/~cjlin/libsvm](http://www.csie.ntu.edu.tw/~cjlin/libsvm). The library provides software for support vector machines for classification and regression and for one-class SVM. Besides, there are available interfaces to LIBSVM for several languages and toolboxes (e.g. R, Python and Perl).

Moreover, software packages based on mathematical toolboxes have been developed. *Kernlab* [44], based on the R toolbox, and can be downloaded from [cran.r-project.org/src/contrib/Descriptions/kernlab.html](http://cran.r-project.org/src/contrib/Descriptions/kernlab.html).

Kernlab provides implementations of support vector machines for classification and regression, gaussian processes, Kernel PCA and spectral clustering algorithms. Finally, *SVM-KMToolbox* [11] is a toolbox, written in MATLAB<sup>®</sup>.<sup>9</sup> It can be downloaded from [asi.insa-rouen.fr/~arakotom/toolbox/index.html](http://asi.insa-rouen.fr/~arakotom/toolbox/index.html) and contains implementations of SVM for classification and regression, multiclass SVM, one-class SVM, kernel PCA and kernel discriminant analysis.

## 9.12 Conclusion

In this chapter we have provided an overview of kernel methods. First of all, we have recalled the basic tools of the optimization theory, the Lagrange multipliers and the Kuhn Tucker theorem, used in the kernel methods. Then support vector machines for classification and regression have been presented. Gaussian processes have been described, underlining their connection with kernel ridge regression. The Fisher kernel discriminant has also been reviewed. Then we have described unsupervised kernel methods, namely kernel PCA and one-class SVM and we have concluded our survey with sketches about kernel and spectral methods for clustering.

Kernel methods are very powerful machine learning algorithms. Nevertheless, their performance is strongly affected by the choice of the appropriate kernel. The choice of the kernel is so important that it has been developed a particular branch of the kernel method theory, called *kernel engineering*, devoted to how to design appropriate kernel for a given task. In the last years, have been designed kernel for image classification [3], for handling word sequences [10], for string and tree matching [35][51][89], for hypertext classification [42]. A detailed discussion on this topic is out of this topic of the book, therefore we advise the reader interested in kernel engineering to refer specific works on kernels such as [78].

Finally, we conclude the chapter providing some bibliographical remarks. SVMs for classification and regression are discussed in detail in [19][74][78][86].

<sup>9</sup> MATLAB<sup>®</sup> is a registered trademark of *The Mathworks, Inc.*

A comprehensive survey of the Gaussian processes is provided by [71]. kernel Fisher discriminant and kernel PCA are described in [75] and [58], respectively. Spectral and kernel methods for clustering are reviewed, underlining their connections, in [29].

## Problems

**9.1.** Consider the function  $K : X \times X \rightarrow \mathbb{R}$ , where  $X \subseteq \mathbb{R}^n$ . Prove that if  $K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$  then  $K(\cdot)$  is a Mercer kernel.

**9.2.** Prove that the *Cauchy kernel*  $C(\mathbf{x}, \mathbf{y}) = \alpha(1 + \|\mathbf{x} - \mathbf{y}\|^2)$  is positive definite for  $\alpha > 0$ . (*Hint:* Read Appendix D).

**9.3.** Prove that the *Epanechnikov kernel*, defined by

$$E(x, y) = 0.75(1 - \|\mathbf{x} - \mathbf{y}\|^2)\mathbf{I}(\|\mathbf{x} - \mathbf{y}\| \leq 1) \quad (9.196)$$

is *conditionally positive definite*. (*Hint:* Read Appendix D).

**9.4.** Prove that the optimal hyperplane is unique.

**9.5.** Consider the SMO algorithm for classification. What is the minimum number of Lagrange multipliers which can be optimized in an iteration? Explain your answer.

**9.6.** Consider the SMO algorithm for classification. Show that in the case of unconstrained maximum we obtain the following updating rule

$$\alpha_2(t+1) = \alpha_2(t) - \frac{y_2(E_1 - E_2)}{2K(\mathbf{x}_1, \mathbf{x}_2) - K(\mathbf{x}_1, \mathbf{x}_1) - K(\mathbf{x}_2, \mathbf{x}_2)} \quad (9.197)$$

where  $E_i = f(\mathbf{x}_i - y_i)$ .

**9.7.** Consider the data Set A of the SantaFe time series competition. Using a public domain SVM regression package and the four preceding values of the time series as input, predict the actual value of the time series. The data set A can be downloaded from <http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html>. Implement a Gaussian process for regression and repeat the exercise replacing SVM with the Gaussian process. Discuss the results.

**9.8.** Using the o-v-r method and a public domain SVM binary classifier (e.g. SVMLight or SVMtorch), test a multiclass SVM on *Iris Data* [31] that can be downloaded by <ftp://ics.uci.edu/pub/machine-learning-databases/iris>. Repeat the same experiment replacing the o-v-r method with the o-v-o strategy. Discuss the results.

**9.9.** Implement kernel PCA and test it on a dataset (e.g. Iris Data). Use as Mercer kernel the Gaussian and verify the Twining and Taylor's result [84], that is, that for large values of the variance the kernel PCA eigenspectrum tends to PCA eigenspectrum.

**9.10.** Consider one-class SVM. Prove there are no bounded support vector when the regularization constant  $C$  is equal to 1.

**9.11.** Implement kernel K-Means and test your implementation on a dataset (e.g. Iris Data). Verify that when you choose as Mercer kernel the inner product you obtain the same results of batch K-Means.

**9.12.**

Implement the Ng-Jordan algorithm using a mathematical toolbox. Test your implementation on Iris data. Compare your results with the ones reported in [9].





---

## References

1. M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, 1964.
2. F. R. Bach and M. I. Jordan. Learning spectral clustering. Technical report, EECS Department, University of California, 2003.
3. A. Barla, E. Franceschi, F. Odone, and F. Verri. Image kernels. In *Proceedings of SVM2002*, pages 83–96, 2002.
4. A. Ben-Hur, D. Horn, H.T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2(2):125–137, 2001.
5. C. Berg, J.P.R. Christensen, and P. Ressel. *Harmonic analysis on semigroups*. Springer-Verlag, 1984.
6. C.M. Bishop. *Neural Networks for Pattern Recognition*. Cambridge University Press, 1995.
7. M. Brand and K. Huang. A unifying theorem for spectral embedding and clustering. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.
8. L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.
9. F. Camastra and A. Verri. A novel kernel method for clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):801–805, 2005.
10. N. Cancedda, E. Gaussier, C. Goutte, and J.-M. Renders. Word-sequence kernels. *Journal of Machine Learning Research*, 3(1):1059–1082, 2003.
11. S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy. SVM and kernel methods matlab toolbox. Technical report, Perception Systemes et Information, INSA de Rouen, 2005.
12. Y. Censor. Row-action methods for huge and sparse systems and their applications. *SIAM Reviews*, 23(4):444–467, 1981.
13. Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Application*, 34(3):321–353, 1981.
14. J.H. Chiang. A new kernel-based fuzzy clustering approach: support vector clustering with cell growing. *IEEE Transactions on Fuzzy Systems*, 11(4):518–527, 2003.

15. R. Collobert and S. Bengio. SVMtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1(2):143–160, 2001.
16. R. Collobert, S. Bengio, and J. Mariethoz. Torch: a modular machine learning software library. Technical report, IDIAP, 2002.
17. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20(3):1–25, 1995.
18. N. Cressie. *Statistics for Spatial Data*. John Wiley, 1993.
19. N. Cristianini, J.S. Taylor, and J. S. Kandola. Spectral kernel methods for clustering. In *Advances in Neural Information Processing Systems 14*, pages 649–655. MIT Press, 2001.
20. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal Royal Statistical Society*, 39(1):1–38, 1977.
21. I.S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the 10<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556. ACM Press, 2004.
22. I.S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (to appear), 2007.
23. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
24. T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2001.
25. P.-H. Fan, R.-E. Chen and C.-J. Lin. Working set selection using the second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.
26. P. Fermat. Methodus ad disquirendam maximam et minimam. In *Oeuvres de Fermat*. MIT Press, 1891 (First Edition 1679).
27. M. Ferris and T. Munson. Interior point method for massive support vector machines. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
28. M. Ferris and T. Munson. Semi-smooth support vector machines. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
29. M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of spectral and kernel methods for clustering. *Pattern Recognition*, to appear.
30. I. Fischer and I. Poland. New methods for spectral clustering. Technical report, IDSIA, 2004.
31. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
32. J. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989.
33. T.T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: a fast and simple learning procedure for support vector machines. In *Proceedings of 15<sup>th</sup> International Conference on Machine Learning*, pages 188–196. Morgan Kaufman Publishers, 1998.
34. K. Fukunaga. *An Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

35. T. Gärtner, J.W. Lloyd, and P.A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.
36. M. Girolami. Mercer kernel based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.
37. F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural network architectures. *Neural Computation*, 7(2):219–269, 1995.
38. J. Hadamard. Sur les problemes aux derivees partielles et leur signification physique. *Bull. Univ. Princeton*, 13:49–52, 1902.
39. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.
40. R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press, 2004.
41. T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods*, pages 169–184. MIT Press, 1999.
42. T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite kernels for hypertext classification. In *Proceedings of the 18<sup>th</sup> International Conference on Machine Learning*, pages 250–257. IEEE Press, 2001.
43. R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. In *Proceedings of the 41<sup>st</sup> Annual Symposium on the Foundation of Computer Science*, pages 367–380. IEEE Press, 2000.
44. A. Karatzoglou, A. Smola, K. Hornik, and A. Zeleis. kernlab- an s4 package for kernel methods in r. *Journal of Statistical Software*, 11(9):1–20, 2004.
45. S. Keerthi, S. Shevde, C. Bhattacharyya, and K. Murthy. Improvements to platt’s smo algorithm for SVM classifier design. Technical report, Department of CSA, Bangalore, India,, 1999.
46. S. Keerthi, S. Shevde, C. Bhattacharyya, and K. Murthy. A fast iterative nearest point algorithm for support vector machine design. *IEEE Transaction on Neural Networks*, 11(1):124–136, 2000.
47. G.A. Korn and T.M. Korn. *Mathematical Handbook for Scientists and Engineers*. Mc Graw-Hill, 1968.
48. H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proceedings of 2<sup>nd</sup> Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 367–380. University of California Press, 1951.
49. J.-L. Lagrange. *Mecanique analytique*. Chez La Veuve Desaint Libraire, 1788.
50. D. Lee. An improved cluster labeling method for support vector clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):461–464, 2005.
51. C. Leslie, E. Eskin, A. Cohen, J. Weston, and A. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.
52. D. Lueberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.
53. D.J.C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.
54. O.L. Mangasarian. Linear and non-linear separation of patterns by linear programming. *Operations Research*, 13(3):444–452, 1965.
55. O.L. Mangasarian and D. Musicant. Lagrangian support vector regression. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, June 2000.
56. G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.
57. M. Meila and J. Shi. Spectral methods for clustering. In *Advances in Neural Information Processing Systems 12*, pages 873–879. MIT Press, 2000.

58. S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.R. Müller. Fisher discriminant analysis with kernels. In *Proceedings of IEEE Neural Networks for Signal Processing Workshop*, pages 41–48. IEEE Press, 2001.
59. M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, 1969.
60. J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.
61. R. Neal. *Bayesian Learning in Neural Networks*. Springer-Verlag, 1996.
62. A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2002.
63. E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII, Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE Press, 1997.
64. E. Osuna and F. Girosi. Reducing the run-time complexity in support vector machines. In *Advances in Kernel Methods*, pages 271–284. MIT Press, 1999.
65. A. Paccanaro, C. Chennubhotla, J.A. Casbon, and M.A.S. Saqi. Spectral clustering of protein sequences. In *Proceedings of International Joint Conference on Neural Networks*, pages 3083–3088. IEEE Press, 2003.
66. J.C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods*, pages 185–208. MIT Press, 1999.
67. J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems 12*, pages 547–553. MIT Press, 2000.
68. T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.
69. M. Pontil and A. Verri. Support vector machines for 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.
70. M.J.D. Powell. Radial basis functions for multivariable interpolation: A review. In *Algorithms for Approximation*, pages 143–167. Clarendon Press, 1987.
71. C.E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
72. K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problem. *Proceedings of the IEEE*, 86(11):2210–2239, 1998.
73. R. Rosipal and M. Girolami. An expectation maximization approach to nonlinear component analysis. *Neural Computation*, 13(3):505–510, 2001.
74. B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.
75. B. Schölkopf, A.J. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.
76. B. Schölkopf, A.J. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical report, Max Planck Institut für Biologische Kybernetik, 1998.
77. B. Schölkopf, R.C. Williamson, A.J. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems 12*, pages 526–532. MIT Press, 2000.
78. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
79. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

80. D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11-13):1191–1199, 1999.
81. A.N. Tikhonov. On solving ill-posed problem and method of regularization. *Dokl. Acad. Nauk USSR*, 153:501–504, 1963.
82. A.N. Tikhonov and V.Y. Arsenin. *Solution of ill-posed problems*. W.H. Winston, 2002.
83. I. Tsochantaris, T. Hoffman, T. Joachims, and Y. Altun. Support vector learning for interdependent and structured output spaces. In *Proceedings of ICML04*. IEEE Press, 2004.
84. C.J. Twining and C.J. Taylor. The use of kernel principal component analysis to model data distributions. *Pattern Recognition*, 36(1):217–227, 2003.
85. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.
86. V.N. Vapnik. *Statistical Learning Theory*. John Wiley, 1998.
87. V.N. Vapnik and A.Ya. Chervonenkis. A note on one class of perceptron. *Automation and Remote Control*, 25:103–109, 1964.
88. V.N. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.
89. S. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Processing Systems 15*, pages 569–576. MIT Press, 2003.
90. U. von Luxburg, M. Belkin, and O. Bosquet. Consistency of spectral clustering. Technical report, Max Planck Institut für Biologische Kybernetik, 2004.
91. U. von Luxburg, M. Belkin, and O. Bosquet. Limits of spectral clustering. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.
92. G. Wahba. *Spline Models for Observational Data*. SIAM, 1990.
93. J. Weston, A. Gammernan, M. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In *Advances in Kernel Methods*, pages 293–306. MIT Press, 1999.
94. J. Weston and C. Watkins. Multi-class support vector machines. In *Proceedings of ESANN99*, pages 219–224. D. Facto Press, 1999.
95. C.K.I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.
96. J. Yang, V. Estvill-Castro, and S.K. Chalup. Support vector clustering through proximity graph modelling. In *Neural Information Processing 2002, ICONIP'02*, pages 898–903, 2002.



## Markovian Models for Sequential Data

### What the reader should know to understand this chapter

- Bayes decision theory (Chapter 5).
- Lagrange multipliers and conditional optimization problems (Chapter 9).
- Probability and statistics (Appendix A).

### What the reader should know after reading this chapter

- The three problems of hidden Markov models.
- The Baum-Welch algorithm.
- The Viterbi algorithm.
- $N$ -gram language modeling.

## 10.1 Introduction

Most of the techniques presented in this book are aimed at making decisions about data. By data it is meant, in general, vectors representing, in some sense, real-world objects that cannot be handled directly by computers. The components of the vectors, the so-called *features*, are supposed to contain enough information to allow a correct decision and to distinguish between different objects (see Chapter 5). The algorithms are typically capable, after a training procedure, of associating input vectors with output decisions. On the other hand, in some cases real-world objects of interest cannot be represented with a single vector because they are sequential in nature. This is the case of speech and handwriting, which can be thought of as sequences of phonemes (see Chapter 2) and letters, respectively, temporal series, biological sequences (e.g. chains of proteins in DNA), natural language sentences, music, etc. The goal of this chapter is to show how some of the techniques presented so far for single vectors can be extended to sequential data.

Given an observation sequence  $S = \mathbf{x}_1^T = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$ , where  $\mathbf{x}_i \in \mathbb{R}^n$  can be continuous or discrete, the problem is to provide a probability density function  $p(S)$  over the space  $\mathcal{S}$  of the sequences. If necessary, the density function must be of the form  $p(S|\Theta)$ , where  $\Theta$  is a parameter set that can be learnt from a training set containing a sufficient number of labeled sequences. This problem has been successfully addressed in the last 20 years using the so-called *probabilistic finite state machines* (PFSM) [37][38], a family of models including probabilistic finite state automata [11], Markov chains [24][29], probabilistic suffix trees [34][33], and other models (see [23] for an extensive survey). This chapter focuses on two particular models of the family, i.e. *N-grams* [31] and *hidden Markov models* (HMMs) [32].

The *N-grams* are simple models giving the probabilities of sequences of elements belonging to a finite alphabet. In particular, the *N-grams* outperform linguistics based approaches in modeling natural sentences [35]. The HMMs are one of the most commonly applied PSFM and have the particularity of modeling sequences of *states* that cannot be observed directly, but only through sequences of statistically related *observations* (see the rest of this chapter for more details). This makes the HMMs more flexible than other models and suitable for problems that cannot be addressed with other kinds of PFSM [7].

The rest of this chapter is organized as follows: Section 10.2 provides the main elements and definitions about HMMs and it explains the reason of the introduction of the nonobservable states, Section 10.3 introduces the three problems characterizing the use of HMMs, i.e. *likelihood*, *decoding* and *learning*, Sections 10.4, 10.5 and 10.6 describe the way such problems are addressed, Section 10.7 presents different variants of the HMMs, Section 10.8 describes the *N-grams* and the data sparseness problem, Section 10.9 introduces discounting and smoothing techniques and Section 10.10 provides a quick tutorial to a free package enabling one to build *N-gram* models.

## 10.2 Hidden Markov Models

In very simple terms, the music can be thought of as a sequence of notes  $S = s_1^T = \{s_1, \dots, s_T\}$  with different durations. The single elements  $s_t$  can be modeled as random variables, called *state variables*, which take values in a finite set  $V = \{v_1, \dots, v_N\}$ <sup>1</sup>, i.e.  $s_t = v_i \forall t \in \{1, \dots, T\}$ , where  $i \in \{1, \dots, N\}$ . Consider the case where the music score is at disposition and the sequence  $S$  can then be accessed directly, the probability  $p(S)$  of the sequence  $S$  being observed can be estimated with a Markov model (MM) of order  $k$ , i.e. a probability distribution defined over sequences and based on the following conditional independence assumption:

<sup>1</sup> The case where  $V$  is a continuous range concerns the so-called *state space models* and it is out of the scope of this book. The interested reader can refer to [7] for more details.



$$p(s_t | s_1^{t-1}) = p(s_t | s_{t-k}^{t-1}), \quad (10.1)$$

i.e. the state variable  $s_t$  depends only on the state variables  $s_{t'}$  with  $t-t' > k$ . In other words, the state variable  $s_t$  depends only on the  $k$  previous state variables in  $S$ . As a consequence, by Equation (10.1) the distribution  $p(S)$  can be decomposed as follows:

$$p(S) = p(s_1^k) \prod_{t=k+1}^T p(s_t | s_{t-k}^{t-1}). \quad (10.2)$$

The correct expression for the fact that  $s_t = v_k$  is *the state variable at step  $t$  takes the value  $v_k$* . However it is more common to say, although not correct, that *the state at step  $t$  is  $v_k$* , and the same convention will be applied throughout this book.

In most cases  $k = 1$  and the above distribution becomes:

$$p(S) = p(s_1) \prod_{t=2}^T p(s_t | s_{t-1}), \quad (10.3)$$

completely specified by the *initial state probabilities*  $p(s_1)$  and by the *transition probabilities*  $p(s_t | s_{t-1})$ . This is the most common case and the assumption that  $k = 1$  is not a restriction because any  $k^{\text{th}}$  order MM can be represented with a first-order model by simply increasing the number of state variables. In fact, if we consider the  $N^k$  sequences  $s_t^{t+k-1}$ , Equation (10.3) can be rewritten as:

$$p(S) = p(s_1^k, s_2^{k+1}, \dots, s_{T-k+1}^T) = p(s_1^k) \prod_{t=2}^{T-k+1} p(s_t^{t+k} | s_{t-1}^{t+k-2}), \quad (10.4)$$

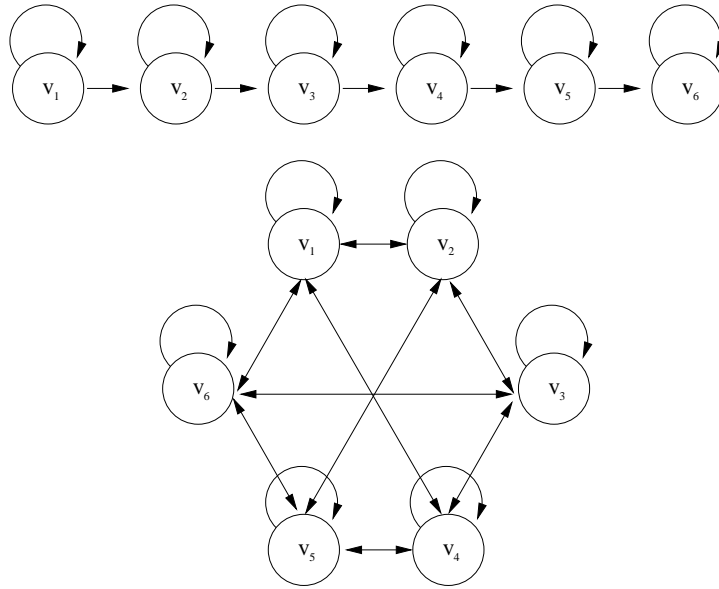
and a  $k^{\text{th}}$  order MM is equivalent to a first-order one.

In principle, the transition probabilities  $p(s_t | s_{t-1})$  depend on  $t$ ; however, this chapter focuses on cases where they are *homogeneous*, i.e. they do not depend on  $t$ . This reduces significantly the number of parameters and enables one to collect all  $p(s_t | s_{t-1})$  into a matrix  $A$ , called a *transition matrix*, such that:

$$a_{ij} = p(s_t = v_j | s_{t-1} = v_i), \quad (10.5)$$

where  $a_{ij}$  is the element  $ij$  of  $A$ . The transition matrix determines the *topology* of the MM, i.e. the structure of the graph that can be used to represent an MM (see Figure 10.1). When  $a_{ij} = 0$ , transitions between states  $v_i$  and  $v_j$  are not allowed and no connection is established between their corresponding nodes. When  $a_{ii} > 0$ , the state  $v_i$  can be repeated in following steps along the sequence and the corresponding transition is called *self-transition*.

When  $a_{ij} > 0$  only for  $j = i$  or  $j = i + 1$ , the model is called *Bakis* (see upper picture in Figure 10.1), when  $a_{ij} > 0$  for  $j \geq i$ , the model topology is called *left-right*. This structure is particularly suitable for data like speech

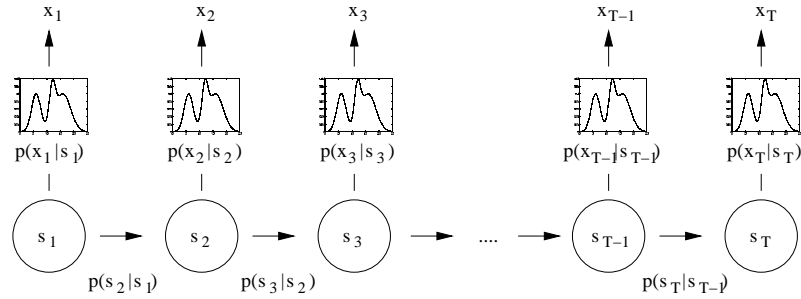


**Fig. 10.1.** Model topology. In the left-right topology (upper figure) only self-transitions and transitions to the next state in a predefined sequence are allowed. In the fully connected model (lower figure) all states can be reached from any other state.

or handwriting where the sequence of states corresponds to a sequence of letters and phonemes, respectively (see Chapters 2 and 12). When  $a_{ij} > 0$ ,  $\forall i, j \in (1, 2, \dots, N)$ , the MM is said to be *fully connected*, and each state can be followed by any other state. A model is said *ergodic* when any state can be reached by any other state in a finite number of steps.

Consider now the case where the music score is not available and the only information at disposition about the music is a recording, i.e. the sequence  $S$  cannot be accessed directly and it is *hidden*. The only possibility of modeling  $p(S)$  is to extract from the sound a vector of measures  $\mathbf{x}_t$  at each time step  $t$  (e.g. the Fourier coefficients described in Appendix B). Since measurement devices are not perfect and the players introduce variations even when they play the same note, the observations  $\mathbf{x}$  corresponding to a specific state  $v_i$  are not constant, but rather follow a distribution  $p(\mathbf{x}|v_i)$  (see Figure 10.2). As a consequence, the sequence  $O = \mathbf{x}_1^T = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$  hardly respects the Markov assumption of Equation (10.1), at least for small  $k$  values. However, the observation sequence  $\mathbf{x}_1^T$  is the effect of the underlying state sequence  $s_1^T$  which respects the Markov assumption, then it is possible to make the following simplifying assumptions:

$$p(\mathbf{x}_t | s_1^T, \mathbf{x}_1^{t-1}) = p(\mathbf{x}_t | s_t) \quad (10.6)$$



**Fig. 10.2.** Hidden Markov models. The figure shows how a sequence of states and observations is generated. The transition between the states is modeled by the transition probabilities  $p(s_{t+1}|s_t)$ , while the observations are generated following the emission probabilities  $p(\mathbf{x}_t|s_t)$ .

$$p(s_t|s_1^{t-1}, \mathbf{x}_1^{t-1}) = p(s_t|s_{t-1}), \quad (10.7)$$

i.e. the probability of the observation  $\mathbf{x}_t$  depends only on state  $s_t$  and this last depends only on state  $s_{t-1}$ . The introduction of the hidden state sequence enables one to model sequences of observations that do not respect directly the Markov assumption, but are generated by *piecewise stationary processes*. In the music example each note lasts for a time interval before the following note is played. During such an interval the sound properties can be assumed to be stationary, i.e. they do not change as much as when passing from one note to the following one, and any form of analysis and measurement produces observations that follow the same distribution.

Based on the independence assumptions of Equations (10.6) and (10.7), the joint distribution of observation and state sequences can be written as follows:

$$p(\mathbf{x}_1^T, s_1^T) = p(s_1) \prod_{t=2}^T p(s_t|s_{t-1}) \prod_{t=1}^T p(\mathbf{x}_t|s_t), \quad (10.8)$$

completely specified by:

- a set  $\boldsymbol{\pi} = \{\pi_1 = p(s_1 = v_1), \dots, \pi_N = p(s_1 = v_N)\}$  of initial state probabilities.
- a transition matrix  $A$  such that  $a_{ij} = p(s_t = v_j|s_{t-1} = v_i)$ .
- a set  $B = \{b_1(\mathbf{x}) = p(\mathbf{x}|v_1), \dots, b_N(\mathbf{x}) = p(\mathbf{x}|v_N)\}$  of emission probability functions.

The set  $\lambda = \{\boldsymbol{\pi}, A, B\}$  is called *hidden Markov model* because the states are not accessible directly, but only through the observations.

### 10.2.1 Emission Probability Functions

The choice of the emission probability function is important because it enables one to distinguish between *discrete* HMMs and *continuous density* (or simply

*continuous*) HMMs. In the first case, the observations belong to a finite set of symbols  $C = \{c_1, c_2, \dots, c_K\}$  and the emission probabilities can be represented with a matrix  $B$  such that:

$$b_{ij} = p(\mathbf{x}_t = c_j | s_t = v_i), \quad (10.9)$$

where  $1 \leq i \leq N$  and  $1 \leq j \leq K$ . Such an approach is especially suitable when the observations are discrete by nature, but it can be used also when the observations are continuous. In fact, it is possible to perform a vector quantization (see Chapter 8) and to replace the observations with their closest codeword. In this way, continuous observations are converted into discrete symbols.

In the case of continuous HMMs, the most common emission probability function is the Gaussian mixture (GM) [9][40]:

$$p(\mathbf{x}_t | s_t = v_i) = \sum_{j=1}^G w_{ij} \frac{1}{\sqrt{2\pi^d |\Sigma_{ij}|}} e^{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_{ij})^T \Sigma_{ij}^{-1} (\mathbf{x} - \boldsymbol{\mu}_{ij})} \quad (10.10)$$

where  $w_{ij}$  is a weight,  $d$  is the dimension of the observation vectors,  $G$  is the number of Gaussians in the mixture,  $\Sigma_{ij}$  is the covariance matrix of the  $j^{\text{th}}$  Gaussian of the mixture corresponding to state  $v_i$  and  $\boldsymbol{\mu}_{ij}$  is the mean for the same Gaussian (see Section 5.7.2 for more details). The *mixture coefficients*  $w_{ij}$  must respect two conditions: the first is that  $w_{ij} > 0 \forall j \in \{1, \dots, G\}$  and the second is that  $\sum_{j=1}^G w_{ij} = 1$ . When  $G = 1$ , the mixture corresponds to a single Gaussian.

Any other continuous distribution can be used, but the GM is the most commonly applied because it has universal approximation properties, i.e. the GM can approximate any other distribution with an error as small as necessary if enough Gaussians are used [30]. On the other hand, the number of Gaussians that can be used is limited by the amount of training material available. In fact, each Gaussian requires  $d^2/2 + 3d/2 + 1$  parameters and the amount of material necessary to train effectively the models grows with the number of parameters.

### 10.3 The Three Problems

The independence assumptions made in Section 10.2 are a key point in the definition of the hidden Markov models. In fact, they enable one to express probability distributions over sequences in terms of a few elements (see Section 10.2): initial state probabilities, transition probabilities and emission probability functions. Such assumptions do not necessarily capture the real relationships between the data under examination (e.g. the music notes in a song), but empirical experience shows that good results are achieved in applications applying the decision theory framework presented in Chapter 5.

In this perspective, there are three problems that must be addressed to use effectively an HMM  $\lambda = \{\boldsymbol{\pi}, A, B\}$ :

**The likelihood problem.** *Given an observation sequence  $O = \mathbf{x}_1^T$  and an HMM  $\lambda = \{\boldsymbol{\pi}, A, B\}$ , how do we estimate the likelihood of  $O$  given  $\lambda$ ?*

The study of this problem leads to the introduction of a trellis allowing one to compute efficiently the quantities necessary to deal not only with the estimation of the likelihood, but also with the other two problems.

**The decoding problem.** *Given an observation sequence  $O = \mathbf{x}_1^T$  and an HMM  $\lambda = \{\boldsymbol{\pi}, A, B\}$ , how do we find the sequence  $S = s_1^T$  that generates  $O$  with the highest probability?*

The examination of this problem leads to the *Viterbi algorithm* (VA), one of the most widely applied decoding approaches.

**The learning problem.** *Given an observation sequence  $O$ , how do we find the model  $\lambda^* = \arg \max_{\lambda} p(O|\lambda)$  that maximizes the likelihood  $p(O|\lambda)$ ?*

The investigation of this problem leads to a particular form of the EM technique (see Chapter 6) known as *Baum Welch* algorithm and is suitable only for the HMMs.

The three problems can be addressed separately and the next subsections describe them in detail.

## 10.4 The Likelihood Problem and the Trellis\*\*

Consider a sequence of observations  $O = \mathbf{x}_1^T$  and a sequence of states  $S = s_1^T$  governed by an HMM  $\lambda$ . The probability of observing the sequence  $O$  when the sequence of states is  $S$  can be written as follows:

$$p(O, S|\lambda) = p(O|S, \lambda)p(S|\lambda). \quad (10.11)$$

The first term of the product can be expressed as:

$$p(O|S, \lambda) = \prod_{t=1}^T b_{s_t}(\mathbf{x}_t), \quad (10.12)$$

and requires only the emission probability functions in  $B$ .

The second term of the product in Equation (10.11) can be estimated using initial state and transition probabilities:

$$p(S|\lambda) = \pi_{s_1} \prod_{t=2}^T a_{s_{t-1}s_t} \quad (10.13)$$

and it requires only the transition probabilities in  $A$ .

The likelihood  $p(O|\lambda)$  corresponds to the probability of Equation (10.11) summed over all possible sequences:

$$p(O|\lambda) = \sum_{S \in \mathcal{S}} p(O|S, \lambda)p(S|\lambda), \quad (10.14)$$

where  $\mathcal{S}$  is the set of all  $T$  long sequences such that  $s_t \in V, \forall t \in \{1, \dots, T\}$ . The number of sequences in  $\mathcal{S}$  is  $N^T$  and, even for moderate values of  $N$  and  $T$ , it is too high to make the explicit computation of  $p(O|\lambda)$  tractable. However, the likelihood can be obtained at a reasonable computational cost by applying a recursive technique based on the *trellis* of Figure 10.3, where each column corresponds to a time step and each node to a state. The links correspond to transitions leading from state  $s_t$  to state  $s_{t+1}$  and to the emission of the observation  $\mathbf{x}_{t+1}$ . No links are allowed between the nodes of the same column because the only allowed transitions are those leading to the next state and observation. A path through the trellis corresponds to a path through the states of an HMM, i.e. to a sequence  $S \in \mathcal{S}$ .

The key element of the technique is the *forward variable*  $\alpha_t(i) = p(\mathbf{x}_1^t, s_t = v_i|\lambda)$ , i.e. the probability of observing the partial sequence  $\mathbf{x}_1^t$  (where  $t \leq T$ ) having  $v_i$  as state  $s_t$ . The forward variable is defined by induction:

**Initialization.** When  $t = 1$ , the forward variable is:

$$\alpha_1(i) = \pi_i b_i(\mathbf{x}_1), \quad (10.15)$$

where  $i = 1, \dots, N$ , and it corresponds to the probability of starting the sequence with the state  $v_i$  and the observation  $\mathbf{x}_1$ .

**Induction.** While the forward variable  $\alpha_1(i)$  is associated to the single node  $i$  in the first column, the forward variable  $\alpha_2(i)$  must take into account all trellis paths starting from the first column and ending at the  $i^{\text{th}}$  node of the second column:

$$\alpha_2(i) = \left[ \sum_{k=1}^N \alpha_1(k) a_{ki} \right] b_i(\mathbf{x}_2) \quad (10.16)$$

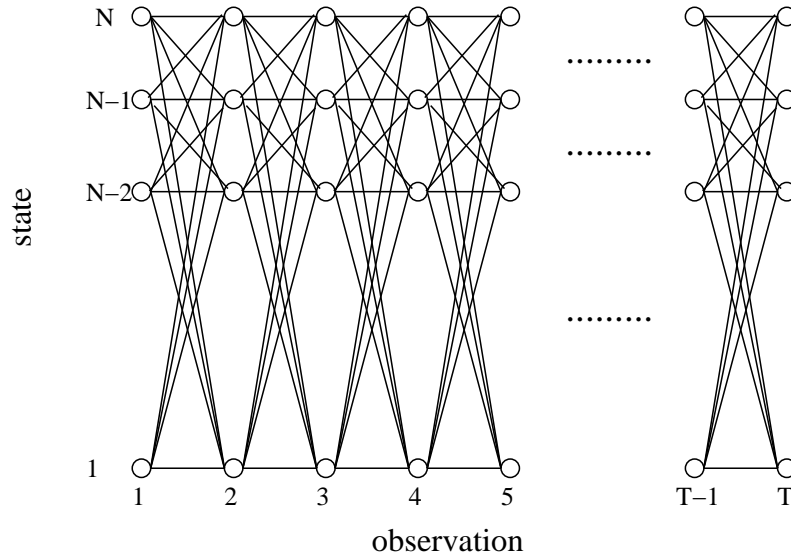
where  $i = 1, \dots, N$ . This corresponds to summing over all links connecting the nodes of the first columns to node  $i$  in the second column. The same consideration made for  $\alpha_2(i)$  applies to the forward variable at any point  $t + 1$  of the sequence:

$$\alpha_{t+1}(i) = \left[ \sum_{k=1}^N \alpha_t(k) a_{ki} \right] b_i(\mathbf{x}_{t+1}), \quad (10.17)$$

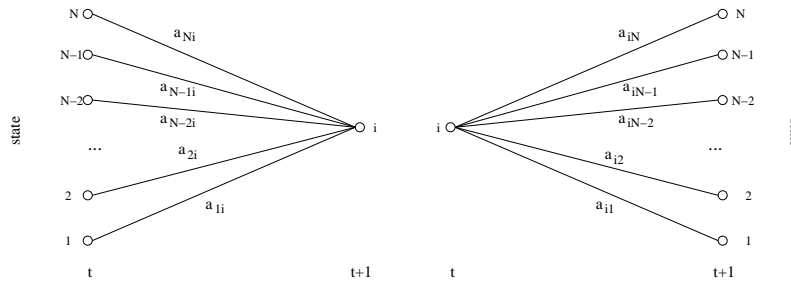
as shown in Figure 10.4 (left plot) where the sum in Equation (10.17) is shown to include all paths leading to state  $v_i$  at step  $t + 1$  in the sequence.

**Termination.** At the last point  $T$ , Equation (10.17) becomes:

$$\alpha_T(i) = \left[ \sum_{k=1}^N \alpha_{T-1}(k) a_{ki} \right] b_i(\mathbf{x}_T), \quad (10.18)$$



**Fig. 10.3.** Trellis. In this lattice, each column corresponds to an observation and each row corresponds to a state. A path through the trellis corresponds to a path through the states of the HMM. The links are associated with the transitions and no links among the elements of the same column are allowed. In fact, each transition must lead to the next state and observation, then to the following column.



**Fig. 10.4.** Forward and backward variables. The left figure shows how the forward variable at point  $t + 1$  of the sequence is obtained by summing over all paths leading from  $s_t$  to  $s_{t+1} = v_i$ . The right figure shows how the backward variable is obtained by summing over all paths starting from state  $s_t = v_i$  and leading to any other state  $s_{t+1}$

and this enables us to write  $p(O|\lambda)$  as follows:

$$p(O|\lambda) = \sum_{i=1}^N \alpha_T(i), \tag{10.19}$$

in fact, this corresponds to the sum over all paths leading to all states at the final sequence point  $T$ .

By applying the above recursive procedure, the number of additions and multiplications is reduced from  $2TN^T$  (the case of the explicit calculation) to  $TN^2$ . In an average handwriting recognition problem (see Chapter 12),  $N$  and  $T$  are around 50 and 100 and the number of operations using the forward variable is around 100 orders of magnitude smaller than the one required by the explicit computation.

## 10.5 The Decoding Problem\*\*

The goal of the decoding is to find the sequence of states  $\hat{S}$  which has the highest probability given an observation sequence  $O$  and an HMM  $\lambda$ :

$$\hat{S} = \arg \max_S p(S|O, \lambda). \quad (10.20)$$

The problem is addressed by applying the *Viterbi algorithm* (VA) [32][39], a *dynamic programming* (DP) [5] based technique using the trellis described above. The main assumption of DP, the so-called *optimality principle*, states that if the path from node  $A$  to node  $C$ , optimal with respect to a given criterion, passes through  $B$ , then also the path from  $B$  to  $C$  is optimal with respect to the same criterion. The VA involves two main operations:

1. To find the estimate of  $p(\hat{S}|O, \lambda)$ , i.e. of the highest probability along a single  $T$ -long path through the states of the HMM.
2. To find the single states  $\hat{s}_1, \hat{s}_2, \dots, \hat{s}_T$  of  $\hat{S}$ .

The first operation relies on the following variable  $\delta_t(i)$ :

$$\delta_t(i) = \max_{s_1^{t-1}} p(s_1^{t-1}, s_t = v_i, \mathbf{x}_1^t | \lambda), \quad (10.21)$$

i.e. on the highest joint conditional probability along a single trellis path for a sequence of  $t$  states terminating with  $v_i$ . The variable  $\delta_t(i)$  is defined by induction.

**Initialization.** When  $t = 1$ , the  $\delta$  variable is:

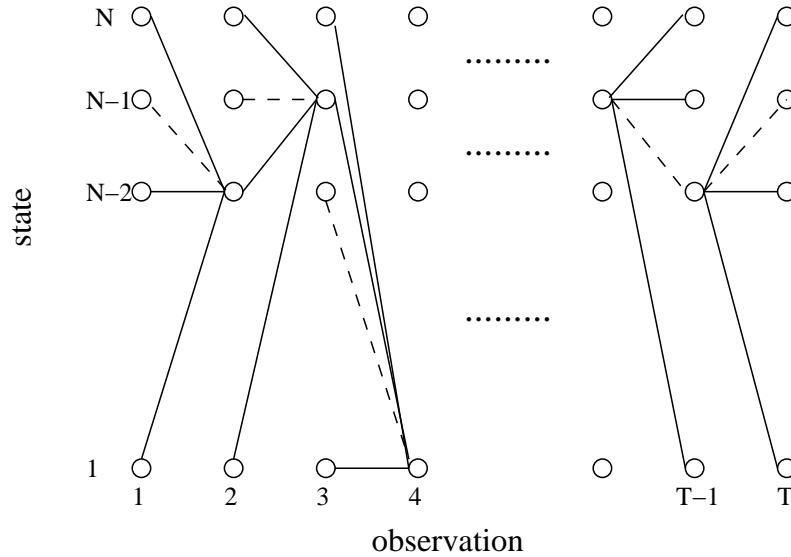
$$\delta_1(i) = \pi_i b_i(\mathbf{x}_1) \quad (10.22)$$

where  $1 \leq i \leq N$ . In other words,  $\delta_1(i)$  corresponds to the probability of starting the state sequence  $\hat{S}$  with  $v_i$  and the value of  $\delta_1(i)$  is associated to the nodes of the first column in Figure 10.5.

**Recursion.** When passing from step  $t$  to step  $t + 1$ , the  $\delta$  variable becomes:

$$\delta_{t+1}(i) = \left[ \max_{k \in (1, \dots, N)} \delta_t(k) a_{ki} \right] b_i(\mathbf{x}_{t+1}) \quad (10.23)$$





**Fig. 10.5.** Viterbi decoding. The dashed paths identified by the  $\delta_t(i)$  variable the connection are associated with the links which lead to a state from the previous time step with the highest probability. The sequence  $\hat{S}$  can be backtracked from the last step ( $t = T$ ) by following the dashed links identified by the  $\psi_t(i)$  variable.

where  $1 \leq i \leq N$  and  $2 \leq t \leq T$ . The rationale behind such a choice can be observed in Figure 10.5. Consider all the paths leading to a specific state  $i$  of the second column, the dashed one maximizes  $\delta_2(i)$ , i.e. it maximizes the following probability:

$$p(s_1^2, \mathbf{x}_1^2 | \lambda) = \pi_{s_1} b_{s_1}(\mathbf{x}_1) a_{s_1 i} b_i(\mathbf{x}_2) \tag{10.24}$$

which is exactly the highest probability for a single path leading to state  $v_i$  at time  $t = 2$ . This is similar to the estimation of the likelihood described in the previous section with the difference that the sum is replaced with a maximization. The same procedure is applied for  $t = 3, 4, \dots, T$  and the result is always the same, i.e. the estimation of the highest probability for a path leading to a certain state at a certain time step.

**Termination.** As a consequence of the recursion procedure, the value of  $\delta_T(i)$  is the highest probability for a trellis path terminating in  $s_T = v_i$ :

$$p(\hat{S}, O | \lambda) = \arg \max_k \delta_T(k), \tag{10.25}$$

and it corresponds to the goal of the first abovementioned operation.

Even if  $p(\hat{S}, O | \lambda)$  is known, the single states  $\hat{s}_1, \dots, \hat{s}_T$  are still unknown and it is necessary to apply a *backtracking* procedure in order to identify them. The backtracking consists in keeping memory of the states which correspond to

the highest probability at each step of the decoding [16]. This is the goal of the second operation mentioned at the beginning of this section. The backtracking can be performed only after the first operation has been completed and it relies on a variable  $\psi_t(i)$  defined by induction.

**Initialization.** For  $t = 1$ , the  $\psi$  variable is:

$$\psi_1(i) = 0, \quad (10.26)$$

where  $1 \leq i \leq N$ .

**Recursion.** The relationship between  $\psi_t(i)$  and  $\psi_{t-1}(k)$  (where  $k = 1, \dots, N$ ) is:

$$\psi_t(i) = \arg \max_{k \in \{1, \dots, N\}} \delta_{t-1}(k) a_{ki} \quad (10.27)$$

where  $1 \leq i \leq N$  and  $2 \leq t \leq T$ . The rationale behind such a choice can be observed in the trellis of Figure 10.5. In the second column, the expression  $\delta_1(k) a_{ki}$  is associated to the edge connecting the node corresponding to state  $v_k$  in the first column to the node corresponding to state  $v_i$  at  $t = 2$ . The link corresponding to the maximum value of such an expression comes from the predecessor at time  $t = 1$  which leads to  $v_i$  at time  $t = 2$  with the highest probability.

**Termination.** The same applies to  $t = 3, 4, \dots, T$  and, at the last column, it is possible to identify the last state of  $\hat{S}$  as follows:

$$\hat{s}_T = \arg \max_{i \in \{1, \dots, N\}} \delta_T(i). \quad (10.28)$$

When the last state of  $\hat{S}$  is known, it is easy to find the other states of the sequence by observing that:

$$\hat{s}_t = \psi_{t+1}(\hat{s}_{t+1}). \quad (10.29)$$

The last expression enables one to backtrack the states of  $\hat{S}$  from  $\hat{s}_T$  to  $\hat{s}_1$  and this is the goal of the second operation as well as of the VA.

The sequence  $\hat{S}$  identified with the VA is *optimal* in the sense of the highest probability criterion (see Equation (10.20)). However, other criteria can lead to other sequences that are optimal under different respects. Although the maximization of  $p(S|O, \lambda)$  is the most commonly applied criterion, it is worth to consider another definition of the optimal sequence, i.e. the sequence  $\hat{S}$  of the states individually most likely:

$$\hat{s}_t = \arg \max_{i=1, \dots, N} p(s_t = v_i | O, \lambda). \quad (10.30)$$

The solution of such a problem requires the definition of a *backward variable*  $\beta_t(i) = p(\mathbf{x}_{t+1}^T | s_t = v_i, \lambda)$  defined by induction.

**Initialization.** When  $t = T$  the backward variable is as follows:

$$\beta_T(i) = 1 \quad (10.31)$$

where  $i = 1, \dots, N$

**Recursion.** If  $t = T - 1$ , then:

$$\beta_{T-1}(i) = p(\mathbf{x}_T | s_{T-1} = v_i, \lambda) = \sum_{k=1}^N a_{ik} b_k(\mathbf{x}_T) \beta_T(k) \quad (10.32)$$

where the factor  $\beta_T(k)$  can be used because it is 1 and it does not modify the result of the sum. When  $t = T - 2$ , the last equation becomes:

$$\beta_{T-2}(i) = p(\mathbf{x}_{T-1}^T | s_{T-2} = v_i, \lambda) = \quad (10.33)$$

$$= \sum_{k=1}^N a_{ik} b_k(\mathbf{x}_{T-1}) \sum_{l=1}^N a_{kl} b_l(\mathbf{x}_T) = \quad (10.34)$$

$$= \sum_{k=1}^N a_{ik} b_k(\mathbf{x}_{T-1}) \beta_{T-1}(k). \quad (10.35)$$

For a generic sequence point  $t$  the backward variable is:

$$\beta_t(i) = \sum_{k=1}^N a_{ik} b_k(\mathbf{x}_{t+1}) \beta_{t+1}(k) \quad (10.36)$$

which corresponds to the right plot in Figure 10.4.

**Termination.** When  $t = 1$ , the backward variable is:

$$\beta_1(i) = \sum_{k=1}^N a_{ik} b_k(\mathbf{x}_2) \beta_2(k) \quad (10.37)$$

where  $i = 1, \dots, N$ .

The product  $\alpha_t(i) \beta_t(i)$  can be transformed into a probability (the demonstration is left for exercise in Problem 10.2):

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{p(O|\lambda)} = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)} \quad (10.38)$$

and the the sequence  $\hat{S}$  can be found as follows:

$$\hat{s}_t = \arg \max_{1 \leq i \leq N} \gamma_t(i). \quad (10.39)$$

The limit of this approach with respect to the VA is evident: since the states at different time steps are considered separately, nothing prevents from finding two states  $v_i$  and  $v_j$  following each other even if  $a_{ij} = 0$ . In other words, since each decision is made at the single state level, global constraints (typically carrying contextual information) are not taken into account with a significant loss of effectiveness.

The solution of the decoding problem leads to a segmentation of the observation sequence where each segment corresponds to a state in the model. As an example consider a handwritten word (see Chapter 12), where the observations are vectors extracted from the word image and the states are letters. When the models work correctly, the decoding splits the vector sequence into segments corresponding to the letters actually composing the handwritten word.

## 10.6 The Learning Problem\*\*

In the previous problems, the parameters of the HMM, i.e. the elements of the set  $\lambda = \{\boldsymbol{\pi}, A, B\}$ , have been considered as given. The subject of the learning problem is how to find such elements and, more in particular, how to estimate them according to state and observation sequences provided for training. No analytical solution is known for this problem and the common approach is to choose  $\lambda^*$  such that:

$$\lambda^* = \arg \max_{\lambda} p(O|\lambda), \quad (10.40)$$

i.e. such that the likelihood  $p(O|\lambda)$  is maximized for the training sequences. This is done through an iterative procedure known as *Baum Welch* [2][3][4] algorithm which is a version of the expectation-maximization technique (see Chapter 6) specific for the HMMs and leads to models maximizing the likelihood over the training data.

In the following  $\lambda^{(i)} = \{\boldsymbol{\pi}^{(i)}, A^{(i)}, B^{(i)}\}$  defines the parameters as estimated at the  $i^{\text{th}}$  iteration. The parameter values at  $i = 0$  are obtained through an initialization procedure (see Section 10.6.1 for different initialization techniques). Given a training observation sequence  $O = \mathbf{x}_1^T$  and the corresponding state sequence  $S = s_1^T$ , the complete data likelihood is  $p(O, S|\lambda)$ . The expected value  $Q(\lambda^{(i)}, \lambda^{(i-1)})$  of the complete-data log-likelihood is then:

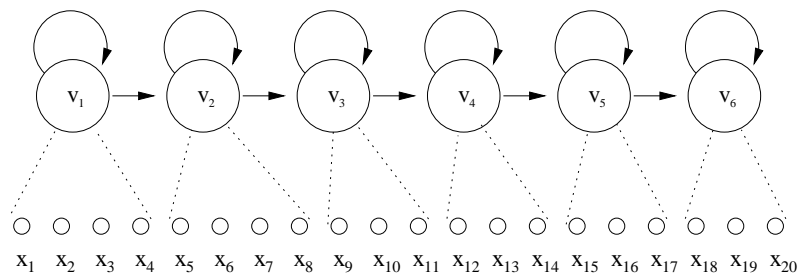
$$Q(\lambda^{(i)}, \lambda^{(i-1)}) = \sum_{S \in \mathcal{S}} \log p(O, S|\lambda^{(i)}) p(O, S|\lambda^{(i-1)}), \quad (10.41)$$

where  $\mathcal{S}$  is the set of all possible  $T$ -long state sequences and  $p(O, S|\lambda^{(i)})$  is:

$$p(O, S|\lambda^{(i)}) = \pi_{s_1}^{(i)} b_{s_1}(\mathbf{x}_1) \prod_{t=2}^T a_{s_{t-1}s_t}^{(i)} b_{s_t}^{(i)}(\mathbf{x}_t). \quad (10.42)$$

By plugging Equation (10.42) into Equation (10.41), the expression of  $Q(\lambda^{(i)}, \lambda^{(i-1)})$  becomes:

$$\begin{aligned} Q(\lambda^{(i)}, \lambda^{(i-1)}) &= \sum_{S \in \mathcal{S}} \log \pi_{s_1}^{(i)} p(O, S|\lambda^{(i-1)}) \\ &+ \sum_{S \in \mathcal{S}} \left( \sum_{t=2}^T \log a_{s_{t-1}s_t}^{(i)} \right) p(O, S|\lambda^{(i-1)}) \\ &+ \sum_{S \in \mathcal{S}} \left( \sum_{t=1}^T \log b_{s_t}^{(i)}(\mathbf{x}_t) \right) p(O, S|\lambda^{(i-1)}) \end{aligned} \quad (10.43)$$



**Fig. 10.6.** Parameter initialization. The emission probability functions are initialized by attributing to each state the observations of an interval. The intervals by partitioning  $O$  into  $N$  intervals as uniform as possible.

where the three parameters ( $\pi$ ,  $A$  and  $B$ ) are split into the three terms of the sum that can be analyzed separately. The EM algorithm can be applied separately to each term leading to the estimation of the parameters the term contains. In the following, Section 10.6.1 shows some initialization techniques, while Section 10.6.2, 10.6.3 and 10.6.4 describe in detail how the three kinds of parameters are estimated.

### 10.6.1 Parameter Initialization

There is no general solution or approach for the initialization problem. The initialization depends on the specific task at hand and on the available a-priori knowledge about the data. There are however two important cases related to the topologies described in Figure 10.1. In the left-right case, the state sequence is typically given a priori, but the number of states is lower than the number of observations, then the decoding aims at attributing a certain number of observations to each state of the sequence (this case will be described better in Chapter 12). The transition probabilities are typically initialized as follows:

$$a_{ij} = \begin{cases} 0.5 & j \in (i, i + 1) \\ 0.0 & j \notin (i, i + 1) \end{cases} \quad (10.44)$$

The emission probabilities are initialized by assigning the same number of observations to each state (the observation sequence is split into  $N$  intervals as uniform as possible) and by estimating means, variances or  $b_{ij}$  values using the observations attributed to each state (see Figure 10.6).

The same approach is applied for the ergodic HMMs with the only difference that the initialization of the transition probabilities is:

$$a_{ij} = \frac{1}{N} \quad \forall i, j \in (1, \dots, N), \quad (10.45)$$

i.e. a uniform distribution.

### 10.6.2 Estimation of the Initial State Probabilities

The first term of the sum in Equation (10.43) involves the initial state probabilities  $\pi_i = p(s_1 = v_i | \lambda)$ . The sum over all sequences  $S \in \mathcal{S}$  can be split into  $N$  sums each involving only the sequences starting with a specific state  $v_j$ :

$$\sum_{S \in \mathcal{S}} \log \pi_{s_1}^{(i)} p(O, S | \lambda^{(i-1)}) = \sum_{k=1}^N \log \pi_k^{(i)} p(O, s_1 = v_k | \lambda^{(i-1)}). \quad (10.46)$$

The estimation can be addressed as a conditional optimization problem (see Section 9.2.1). Adding the Lagrange multiplier  $\gamma$ , using the constraint  $\sum_{l=1}^N \pi_l^{(i)} = 1$  and setting the derivative equal to zero, the result is:

$$\frac{\partial}{\partial \pi_k^{(i)}} \left[ \sum_{l=1}^N \log \pi_l^{(i)} p(O, s_1 = v_l | \lambda^{(i-1)}) - \gamma \left( \sum_{l=1}^N \pi_l^{(i)} - 1 \right) \right] = 0 \quad (10.47)$$

which leads to:

$$\gamma = \sum_{k=1}^N p(O, s_1 = v_k | \lambda^{(i-1)}) = p(O | \lambda^{(i-1)}) \quad (10.48)$$

$$\pi_k^{(i)} = \frac{p(O, s_1 = v_k | \lambda^{(i-1)})}{p(O | \lambda^{(i-1)})}. \quad (10.49)$$

The above expression shows that the  $\pi_k^{(i)}$  estimate is nothing else than the fraction between the likelihood of  $O$  when the state sequences start with  $v_k$  and the likelihood of  $O$  without any constraint. This is reasonable because such a quantity corresponds to the expected fraction of times  $v_k$  is the initial state of a sequence for the observation sequence  $O$ .

The  $\pi_k^{(i)}$  values can be computed efficiently using the  $\gamma_t(k)$  variables introduced in the previous part of this chapter (see Equation (10.38)), in fact:

$$\pi_k^{(i)} = \gamma_1(k) \quad (10.50)$$

(the demonstration is the subject of Problem 10.4).

### 10.6.3 Estimation of the Transition Probabilities

The transition probabilities appear in the second term of Equation (10.43). Using the same approach as in Section 10.6.2, the estimates of  $a_{mn}^{(k)}$  can be obtained as the solutions of the following equations:

$$\frac{\partial}{\partial a_{mn}^{(k)}} \left[ \sum_{S \in \mathcal{S}} \left( \sum_{t=2}^T \log a_{s_{t-1} s_t}^{(k)} \right) p(O, S | \lambda^{(k-1)}) - \gamma \left( \sum_{j=1}^N a_{ij}^{(k)} - 1 \right) \right] = 0, \quad (10.51)$$

where  $m, n = 1, \dots, N$ , which can be rewritten as:

$$\frac{\partial}{\partial a_{mn}^{(k)}} \left[ \sum_{i=1}^N \sum_{j=1}^N \sum_{t=2}^T \log a_{ij}^{(k)} p(O, s_{t-1} = v_i, s_t = v_j | \lambda^{(k-1)}) - \gamma (\sum_{j=1}^N a_{ij}^{(k)} - 1) \right] = 0. \quad (10.52)$$

The result of the above equation is:

$$\gamma = \sum_{t=2}^T \sum_{n=1}^N p(O, s_{t-1} = v_m, s_t = v_n | \lambda^{(k-1)}) = \sum_{t=2}^T p(O, s_{t-1} = v_m | \lambda^{(k-1)}) \quad (10.53)$$

and:

$$a_{mn}^{(k)} = \frac{\sum_{t=2}^T p(O, s_{t-1} = v_m, s_t = v_n | \lambda^{(k-1)})}{\sum_{t=2}^T p(O, s_{t-1} = v_m | \lambda^{(k-1)})}. \quad (10.54)$$

Like in the case of the initial state probabilities, it is possible to obtain  $a_{mn}^{(k)}$  efficiently by using the variables defined in the previous sections:

$$a_{mn}^{(k)} = \frac{\sum_{t=2}^{T-1} \xi_t(m, n)}{\sum_{t=2}^{T-1} \gamma_t(m)}. \quad (10.55)$$

where  $\gamma_t(m)$  has been defined in Section 10.6.2 and the new variable  $\xi_t(i, j)$  is:

$$\xi_t(m, n) = \frac{\sum_{t=2}^T p(O, s_{t-1} = v_m, s_t = v_n | \lambda^{(k-1)})}{p(O | \lambda^{(k-1)})}, \quad (10.56)$$

and can be obtained using  $\alpha_t(i)$  and  $\beta_t(i)$ :

$$\xi_t(m, n) = \frac{\alpha_t(m) a_{mn} b_n(\mathbf{x}_{t+1}) \beta_{t+1}(n)}{p(O | \lambda)}. \quad (10.57)$$

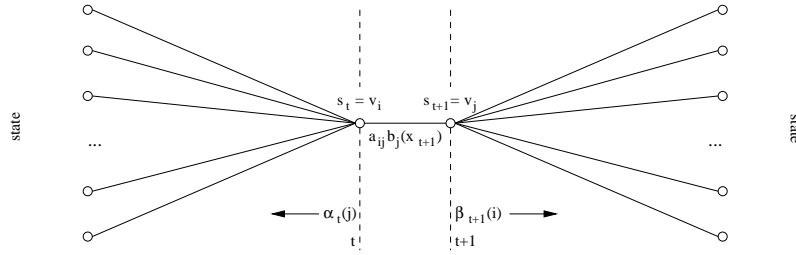
(the demonstration is the goal of Problem 10.5). The variable  $\xi_t(i, j)$  is the probability of a transition between states  $i$  and  $j$  at step  $t$  and it is illustrated in Figure 10.7. In fact, the plot shows that the  $\alpha_t(i)$  variable provides the probability of a path passing through state  $v_i$  at time  $t$ , the product  $a_{ij} b_j(\mathbf{x}_t)$  is the probability of moving from  $v_i$  to  $v_j$  and that  $\beta_t(j)$  gives the probability of the rest of state and observation sequences. The product of the three above probabilities is then the probability of having a transition between  $v_i$  and  $v_j$  at point  $t$  in the sequence.

#### 10.6.4 Emission Probability Function Parameters Estimation

The emission probability functions are included in the third, and last, term of Equation (10.43). The term can be written as follows:

$$\sum_{S \in \mathcal{S}} \sum_{t=1}^T \log b_{s_t}^{(i)}(\mathbf{x}_t) p(O, S | \lambda^{(i-1)}) = \sum_{j=1}^N \sum_{t=1}^T \log b_j^{(i)}(\mathbf{x}_t) p(O, s_t = v_j | \lambda^{(i-1)}) \quad (10.58)$$

and it is necessary to consider separately two cases:



**Fig. 10.7.**  $\xi$  variable. The picture shows how the  $\xi_t(i, j)$  variable accounts for the transition from state  $v_i$  to state  $v_j$  at time  $t$ .

- The HMM is *discrete*, then  $x_t \in C = \{c_1, \dots, c_K\}$  and the emission probabilities are arranged in a matrix  $B$  such that  $b_{ij} = b_i(c_j)$ .
- The HMM is *continuous density* and the expression of  $b_i(\mathbf{x})$  depends on the specific probability density function selected. Since it is the most common case, this section considers the Gaussian mixture (GM), i.e.  $b_i(\mathbf{x}) = \sum_k w_{ik} \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_{ik}, \Sigma_{ik})$ .

In the first case, the values of  $b_{ij}$  maximizing the likelihood can be found by solving the following equation:

$$\frac{\partial}{\partial b_{kl}^{(i)}} \left[ \sum_{j=1}^N \sum_{t=1}^T \log b_{j\mathbf{x}_t}^{(i)} p(O, s_t = v_j | \lambda^{(i-1)}) - \gamma \left( \sum_{n=1}^K b_{kn}^{(i)} - 1 \right) \right] \quad (10.59)$$

where  $\gamma$  is the Lagrange multiplier and the constraint that  $\sum_{n=1}^K b_{kn} = 1$  is used. The solution of this equation is:

$$b_{ij}^{(i)} = \frac{\sum_{t=1}^T p(O, s_t = v_i | \lambda^{(i)}) \delta_{\mathbf{x}_t c_j}}{\sum_{t=1}^T p(O, s_t = v_i | \lambda^{(i)})} = \frac{\sum_{t=1}^T \gamma_t(i) \delta_{\mathbf{x}_t c_j}}{\sum_{t=1}^T \gamma_t(i)} \quad (10.60)$$

where  $\delta_{\mathbf{x}_t c_j}$  is the Kronecher delta ( $\delta_{kl} = 1$  if  $k = l$  and 0 otherwise), in fact only the observations equal to  $c_j$  contribute to the numerator sum. The use of  $\gamma_t(i)$  enables to compute the estimates  $b_{ij}^{(i)}$  efficiently and shows that they are obtained as the fraction between the number of times observation  $c_j$  is emitted when the state is  $v_i$ , and the number of times the state sequences pass through state  $v_i$ .

The second case requires a change in the expression of the complete data likelihood that results in a different function  $Q(\lambda^{(i)}, \lambda^{(i-1)})$ . In fact, the complete-data involves not only the sequence of states underlying the sequence of observations, but also a sequence  $M = \{m_{s_1,1}, \dots, m_{s_T,T}\}$  that contains, at each step  $t$ , the Gaussian  $m_{s_t,t}$  in the mixture corresponding to state  $s_t$  responsible for the emission of  $\mathbf{x}_t$ . The consequence is that  $Q(\lambda^{(i)}, \lambda^{(i-1)})$  modifies as follows:



$$Q(\lambda^{(i)}, \lambda^{(i-1)}) = \sum_{S \in \mathcal{S}} \sum_{M \in \mathcal{M}} \log p(O, S, M | \lambda^{(i)}) p(O, S, M | \lambda^{(i-1)}). \quad (10.61)$$

Fortunately, this affects only the third term of Equation (10.43), thus nothing changes for what concerns initial state and transition probabilities and the results of Section 10.6.2 and Section 10.6.3 apply also in this case.

The third term of Equation (10.43) becomes:

$$\begin{aligned} & \sum_{S \in \mathcal{S}} \left( \sum_{t=1}^T \log b_{s_t}^{(i)}(\mathbf{x}_t) \right) p(O, S | \lambda^{(i-1)}) = \\ & = \sum_{i=1}^N \sum_{l=1}^G \sum_{t=1}^T \log(w_{il}^{(i)} \mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{il}^{(i)}, \Sigma_{il}^{(i)})) p(O, s_t = v_i, m_{s_t, t} = l | \lambda^{(i-1)}) \end{aligned} \quad (10.62)$$

Following the same approach used to estimate the other parameters (derivation with respect to  $w_{kn}^{(i)}$  and use of the Lagrange multiplier with the constraint  $\sum_{l=1}^G w_{kl}^{(i)} = 1$ ) the above equation leads to the following expression for the  $w_{kn}^{(i)}$  coefficients:

$$w_{kn}^{(i)} = \frac{\sum_t p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)})}{\sum_{l=1}^G \sum_{t=1}^T p(O, s_t = v_k, m_{s_t, t} = l | \lambda^{(i-1)})}. \quad (10.63)$$

Also in this case it is possible to use the  $\gamma_t(i)$  variables introduced in the previous part of the chapter to perform an efficient computation of the parameter estimates. The mixture coefficients can be written as follows:

$$w_{kn}^{(i)} = \frac{\sum_{t=1}^T \gamma_{tn}(i)}{\sum_{t=1}^T \gamma_t(i)} \quad (10.64)$$

where

$$\gamma_{tn}(i) = \gamma_t(i) \frac{w_{il}^{(i)} \mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{il}^{(i)}, \Sigma_{il}^{(i)})}{\sum_{l=1}^G w_{il}^{(i)} \mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{il}^{(i)}, \Sigma_{il}^{(i)})}. \quad (10.65)$$

The demonstration of Equation (10.64) is the subject of Problem 10.3.

Equation (10.62) can now be derived with respect to  $\boldsymbol{\mu}_{kn}$  and posed equal to zero in order to estimate the means of the Gaussians. The resulting equation is:

$$\sum_{t=1}^T \Sigma_{kn}^{(i)} (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)}) p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)}) = 0 \quad (10.66)$$

(The demonstration is the goal of Problem 10.6) and its solution is:

$$\boldsymbol{\mu}_{kn}^{(i)} = \frac{\sum_{t=1}^T \mathbf{x}_t p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)})}{\sum_{t=1}^T p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)})}, \quad (10.67)$$

in terms of the  $\gamma_{tn}(k)$  variables, the above corresponds to:

$$\boldsymbol{\mu}_{kn}^{(i)} = \frac{\sum_{t=1}^T \mathbf{x}_t \gamma_{tn}(k)}{\sum_{t=1}^T \gamma_{tn}(k)}, \quad (10.68)$$

i.e. the average observation vector when the state is  $k$  and the observation is emitted by the  $n^{\text{th}}$  Gaussian of the corresponding mixture.

The last parameters to estimate are the covariance matrices  $\Sigma_{ij}^{(i)}$  which can be obtained, as usual, by finding the  $\Sigma_{kn}^{(i)}$  values such that the derivative of the complete-data likelihood in Equation (10.62) is zero:

$$\begin{aligned} & \sum_{t=1}^T \log(|\Sigma_{kn}^{(i)}|) p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)}) + \\ & + \sum_{t=1}^T (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)})^T (\Sigma_{kn}^{(i)})^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)}) p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)}) = 0, \end{aligned} \quad (10.69)$$

the above expression is obtained by taking into account that, if  $A$  is a matrix,  $d \log(|A|)/dA = 2A^{-1} - \text{diag}(A^{-1})$  and  $d(\mathbf{x}^T A \mathbf{x})/dA = (A + A^T)\mathbf{x}$ . The solution of the above equation is as follows:

$$\Sigma_{kn}^{(i)} = \frac{\sum_{t=1}^T (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)})^T (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)}) p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)})}{\sum_{t=1}^T p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)})}, \quad (10.70)$$

and it can be computed efficiently in the following way:

$$\Sigma_{kn}^{(i)} = \frac{\sum_{t=1}^T (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)})^T (\mathbf{x}_t - \boldsymbol{\mu}_{kn}^{(i)}) \gamma_{tn}(k)}{\sum_{t=1}^T \gamma_{tn}(k)}, \quad (10.71)$$

i.e. by finding the covariances of the observation components when the state is  $v_k$  and the observations are emitted by the  $n^{\text{th}}$  Gaussian of the mixture.

## 10.7 HMM Variants

The HMM architecture presented so far is the most commonly applied and it has been shown to be effective in a wide spectrum of problems. However, there are some specific domains that require some variations or adaptations for the HMMs to be as effective as in other cases. This section presents a quick, and not exhaustive, survey of the major HMM variants presented in the literature. More extensive surveys can be found in [7][32] for general aspects, in [17] for control applications, in [10] for econometrics, and in [1][28] for bioinformatics.

Section 10.2 introduces the homogeneity assumption, i.e. the fact that the transition probabilities do not depend on the step  $t$  of the state sequence. The input-output HMMs (IOHMM) [8] remove such a hypothesis and condition transition and emission probabilities to an *input sequence*  $Y = \mathbf{y}_1^L$ , where  $L$  (the input sequence length) is not necessarily equal to  $T$  (the state and observation sequences length). This means that an IOHMM is not a probability distribution  $p(\mathbf{x}_1^T)$  defined over the space of the observation sequences, but rather a conditional distribution  $p(\mathbf{x}_1^T | \mathbf{y}_1^L)$ . In the simpler case,  $L = T$  and the theory of the IOHMMs is close to that of the HMMs. In fact, transition probabilities  $p(s_t | s_{t-1})$  and emission probabilities  $p(\mathbf{x}_t | s_t)$  are simply replaced

with conditional probabilities of the form  $p(s_t|s_{t-1}, \mathbf{y}_t)$  and  $p(\mathbf{x}_t|s_t, \mathbf{y}_t)$  respectively. In more general terms, transition and emission probabilities can be conditioned to a subsequence  $\mathbf{y}_{t-K}^{t+K}$  of the input sequence. IOHMMs have been applied in control theory (where they are called *partially observable Markov decision processes* [7]). Their goal is to find a sequence of actions (taken as an input sequence) minimizing a cost function defined over the sequences of the observed outputs. In this case, the IOHMM represents the probabilistic relationship between actions and effects with an hidden state variable.

Based on the fact that HMMs and artificial neural networks (ANN) have complementary properties, several approaches tried to combine the two algorithms resulting into the so-called *hybrid HMM-artificial neural networks models* [11]. HMMs are suitable for sequential data, but they make assumptions about the distribution of the data. On the other hand, ANNs can approximate any kind of nonlinear discriminant functions and do not make assumptions about the data, but they are not made for handling sequential data. One possible combination approach is to train the ANN in order to provide the *a-posteriori* probability of a state given an observation. In other words, if  $g_k(\mathbf{x}_t|\Theta)$  is the  $k^{\text{th}}$  output of an ANN, typically a multilayer perceptron (see Chapter 8), then:

$$g_k(\mathbf{x}_t|\Theta) \simeq p(s_t = v_k|\mathbf{x}_t), \quad (10.72)$$

where  $\Theta$  is the parameter set of the neural network. The use of such an approach for sequence recognition (with related training and decoding algorithms) is illustrated in [12][13]. Another combination approach consists in turning local posterior probabilities into *scaled likelihoods* defined as follows:

$$\frac{p(s_t = v_k|\mathbf{x}_t)}{p(s_t = v_k)} = \frac{p(\mathbf{x}_t|s_t = v_k)}{p(\mathbf{x}_t)} \quad (10.73)$$

where the prior  $p(s_t = v_k)$  of state  $v_k$  can be estimated using the frequency it has in the data and  $p(\mathbf{x}_t)$  is state independent and is simply a normalization constant that does not need to be estimated. The advantage of this combination approach is that the scaled likelihoods are trained discriminatively (thanks to the ANN properties) and can be used in a Viterbi Algorithm to estimate the global scaled likelihood [22]:

$$\frac{p(O|S, \Theta)}{p(O)} = \sum_{S \in \mathcal{S}} \prod_{t=2}^T \frac{p(\mathbf{x}_t|s_t = v_k)}{p(\mathbf{x}_t)} p(s_t|s_{t-1}). \quad (10.74)$$

These hybrid HMM/ANN approaches provide more discriminant estimates of the emission probabilities without requiring strong hypotheses about the statistical distribution of the data.

Some problems require the joint modeling of two sequences  $\mathbf{x}_1^T$  and  $\mathbf{y}_1^L$  of different length. This is typical in multimodal data processing where different streams of information are extracted from the same events but with different sampling rates. A typical example are videos where the visual sampling rate

is 24 images per second, while the audio rate is 8000 samples per second. A recently proposed approach [6] uses two hidden variables to account for such a situation. The first is a common state variable  $s_t$  and it is associated to the longest sequence. The second is a *synchronization variable*  $\tau_t$  which accounts for the alignment between the two sequences. This means that the asynchronous HMM models the distribution  $p(\mathbf{x}_1^T, \mathbf{y}_1^L, s_1^T, \tau_1^T)$ , where  $T > L$ . The resulting model is called *asynchronous HMM* and it can be trained with an appropriate EM algorithm [6].

In some other cases, the models are required to account for non-stationary changes in the process underlying the observation production. This is especially needed in econometric models of market changes due to unexpected events [18][20][36]. A common approach is to use a regression model:

$$\mathbf{x}_t = \beta_{s_t} \mathbf{y}_t + \epsilon_t \quad (10.75)$$

where  $\mathbf{x}_t$  is the observation at time  $t$ ,  $\epsilon_t$  is a random variable with zero-mean Gaussian distribution,  $\mathbf{y}_t$  is an input sequence, and  $\beta_{s_t}$  is a set of parameters depending on the discrete state variable  $s_t$ . This specifies a particular form of  $p(\mathbf{x}_t | \mathbf{y}_t, s_t)$  for an IOHMM (see above). The joint distribution of  $\mathbf{x}_1^T$  and  $s_1^T$  requires to model also the state variable. This is typically done through a transition probability matrix as in the common HMMs.

Another interesting problem is the use of a continuous state variable which leads to the so-called *state-space models* (SSM). Most SSMs are based on transition probabilities of the following kind:

$$p(s_t | s_{t-1}, \mathbf{x}_t) = \mathcal{N}(s_t, A s_{t-1} + B \mathbf{x}_t, \Sigma) \quad (10.76)$$

i.e. Gaussians distributions where the average is a linear function ( $A$  and  $B$  are matrices) of previous state and current observation, a choice motivated mainly by tractability problems [7]. The *Kalman Filter* corresponds to such a model [25].

## 10.8 $N$ -gram Models and Statistical Language Modeling

The  $N$ -gram models, or  $N$ -grams *tout court*, are a simple and effective approach to estimate the probability of sequences containing symbols belonging to a finite alphabet. The  $N$ -grams can be used for any kind of sequence, but their most successful application is the modeling of word sequences in natural language texts. In fact, even if they do not involve any linguistic knowledge, the  $N$ -grams achieve state of the art performances in language modeling [35] and are widely applied in speech and handwriting recognition systems (see Chapter 12). After a general description of the  $N$ -grams, the next sections present the main problems associated to them: the estimation of the parameters and the data *sparseness*, including the necessity of *smoothing* and *unseen events* probability estimation. The use of the *SLM-CMU toolkit*, a free software package enabling one to build  $N$ -gram models, will be illustrated at the end of this part.

### 10.8.1 $N$ -gram Models

Consider a finite alphabet  $T = \{t_1, t_2, \dots, t_M\}$  containing  $M$  symbols  $t_i$  and a sequence  $W = w_1^L$ , where  $w_i \in T \forall i \in (1, \dots, L)$ .  $W$  can be assumed as the product of a Markov source, then the probability  $p(W)$  of observing  $W$  can be written as follows:

$$p(W) = p(w_1) \prod_{i=2}^L p(w_i | w_1^{i-1}), \quad (10.77)$$

where the sequence  $w_1^{i-1}$  is called *history*  $h_i$  of  $w_i$ . The number of possible histories of  $w_i$  is  $M^{i-1}$ , a quantity that becomes rapidly high even for moderate values of  $M$  and  $i$ . This can create severe problems in estimating the probabilities of Equation (10.77). In fact, reliable estimates of  $p(w_i | w_1^{i-1})$  can be obtained only if each  $w_1^i$  is represented a sufficient number of times in the training data. On the other hand, if the number of possible sequences is high, it can be difficult to collect enough training material.

One possible solution is to group all histories  $h_i = w_1^{i-1}$  into classes of equivalence  $\Phi(h_i) : T^{i-1} \rightarrow C$ , where  $C = (1, \dots, K)$  is a set of classes containing  $K \ll M^{i-1}$  elements. This changes Equation (10.77) into the following expression:

$$p(W) = p(w_1) \prod_{i=2}^L p(w_i | \Phi(h_i)). \quad (10.78)$$

The form of  $\Phi(h_i)$  depends on the specific application and it can involve domain specific knowledge. However, a common and general approach is to group all histories ending with the same  $N - 1$  symbols:

$$p(W) = p(w_1) \prod_{i=2}^L p(w_i | \Phi(h_i)) = p(w_1^{N-1}) \prod_{i=N}^L p(w_i | w_{i-N+1}^{i-1}), \quad (10.79)$$

this corresponds to a Markov Model of order  $N$  (see Section 10.2) and this is what is called an  $N$ -gram model. Depending on the value of  $M$ , the number of equivalence classes can still grow quickly with  $N$  and, in practice, orders higher than three are rarely used. The problem of  $p(w_1^{N-1})$  can be solved in different ways. A sequence of  $N - 1$  null symbols can be added before  $w_1$  so that an  $N - 1$  long history is present also for  $w_i$  with  $i < N$ . Another solution is to use histories with less than  $N - 1$  elements when  $i < N$ .

### 10.8.2 The Perplexity

The *perplexity* is the performance measure used to assess the quality of the  $N$ -gram models. Given a test sequence  $W = w_1^L$ , not included in the training corpus, the perplexity is obtained as follows:

$$PP = \left[ \frac{1}{p(W)} \right]^L = \left[ \prod_{i=1}^L p(w_i|h_i) \right]^{-\frac{1}{L}}. \quad (10.80)$$

The rationale behind such an expression can be understood by considering the expression of  $\log PP$ :

$$\log PP = -\frac{1}{L} \sum_{i=1}^L \log[p(w_i|h_i)], \quad (10.81)$$

the logarithm of the perplexity is the opposite of the average of the logarithm of  $p(w_i|h_i)$ . When the probability  $p(w|h_i)$  is high, it means that the model is capable of predicting with high probability the symbols actually appearing in the test sequence. Since higher values of  $p(w|h)$  result into lower values of  $-\log[p(w|h)]$ , lower  $PP$  correspond to better models, i.e. to models where the average  $p(w|h)$  is higher. In other words, the lower the perplexity, the better the model.

If the distribution  $p(w|h)$  is uniform, i.e.  $p(w|h) = 1/M \forall w \in T$ , where  $M$  is the size of the lexicon, then the perplexity achieves the highest possible value. In fact,  $p(w|h) = 1/M$  is the solution of the following equation:

$$\frac{\partial}{\partial p(w|h)} \left[ -\frac{1}{L} \sum_{i=1}^L \log p(w_i|h_i) + \lambda \left( \sum_{w'} p(w'|h) - 1 \right) \right] = 0 \quad (10.82)$$

where  $\lambda$  is a Lagrange multiplier.

When  $p(w|h)$  is uniform, the *average branching factor*, i.e. the average number of symbols with probability significantly higher than zero, of the  $N$ -gram model is  $M$ . This provides a further interpretation of the perplexity as the average branching factor of the model. If the perplexity is small compared to  $M$ , it means that most of the symbols in the dictionary are discarded by the model and viceversa. On the other hand, it is not guaranteed that only wrong symbols are discarded, then the perplexity is not always representative of the actual performance of the model [27]. However, although such a problem, the PP is widely applied in the literature and it provides reasonable estimates of the models performance.

### 10.8.3 $N$ -grams Parameter Estimation

The probabilities  $p(w_i|w_{i-N+1}^{i-1})$  are the parameters of the  $N$ -gram models and can be estimated by maximizing the likelihood over a training set of sequences. The training set can be thought of as a single sequence  $W = w_1^L$  and the likelihood of the model can be written as follows:

$$p(W|\{p(w|h)\}) = \prod_{i=1}^L p(w_i|h_i), \quad (10.83)$$

where  $\{p(w|h)\}$  is the set of all possible probabilities  $p(w|h)$ , i.e. the parameters set of the  $N$ -gram model. The loglikelihood corresponds to the following expression:

$$\log p(W|\{p(w|h)\}) = \sum_{i=1}^L \log p(w_i|h_i) = \sum_{h \in \mathcal{H}} \sum_{w \in T} N(w, h) \log p(w|h), \quad (10.84)$$

where  $\mathcal{H}$  is the set of all possible histories,  $T$  is the set of all possible symbols and  $N(w, h)$  is the number of times the event  $(h, w)$ , i.e. history  $h$  followed by symbol  $w$ , has been observed in the training set. The estimates of  $p(w|h)$  maximizing the likelihood can be found as the solutions of the following equations (the estimation is addressed as a conditional optimization problem using the approach described in Section 9.2.1):

$$\frac{\partial}{\partial p(w|h)} \left[ \sum_{h' \in \mathcal{H}} \sum_{w' \in T} N(w', h') \log p(w'|h') - \sum_{h' \in \mathcal{H}} \mu_{h'} \left( \sum_{w' \in T} p(w'|h') - 1 \right) \right] = 0 \quad (10.85)$$

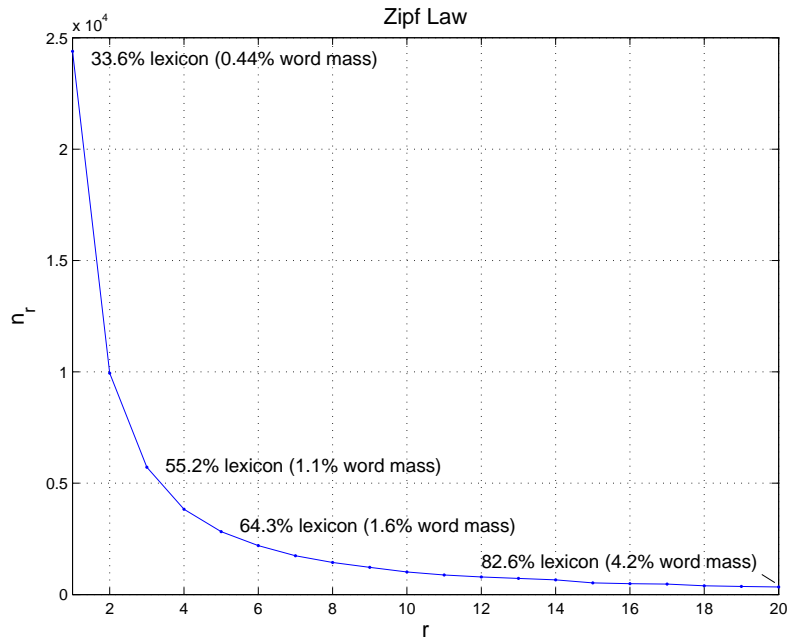
where  $\mu_{h'}$  is a Lagrange multiplier and the constraint  $\sum_{w \in T} p(w|h) = 1$  is used. The solution of such an equation is:

$$p(w|h) = \frac{N(w, h)}{\sum_{w' \in T} N(w', h)} = \frac{N(w, h)}{N(h)} \quad (10.86)$$

i.e. the parameters correspond to the relative frequencies of sequences  $(h, w)$  with respect to sequences  $h$ . The ML training of the  $N$ -gram model can then be performed by simply counting the number of times symbol sequences of length  $N$  appear in the training data. This is very simple, but it leaves open the problem of the  $N$  long sequences that do not appear in the training set. Moreover, it makes an event appearing two times twice as probable as an event appearing only once. This is not correct because such small differences in  $N(w, h)$  are likely to be caused by random fluctuations. These problems are inherent to data sparseness, a phenomenon affecting many  $N$ -gram applications and explained in the next section in the case of natural language texts.

#### 10.8.4 The Sparseness Problem and the Language Case

The maximum-likelihood estimation of the probabilities  $p(w|h)$  relies on the hypothesis that all events  $(h, w)$  are sufficiently represented in the training set. However, such an hypothesis is unrealistic in most cases and data tend rather to be *sparse*, i.e. to contain a high percentage of *singletons* (the events appearing only once). Moreover, it happens often that the number of possible sequences  $(h, w)$  is so high that it is not possible to find a training set containing all possible events. This poses two main problems: the first is the *smoothing* of the probabilities estimated for rare events. The second is the estimation of



**Fig. 10.8.** The Zipf law. The plot shows the number of words appearing  $r$  times as a function of the  $r$  position in the ranking of the represented  $r$  values.

the probability of *unseen* events. Both problems will be addressed in the rest of this chapter, but this section provides some insights on the sparseness phenomenon by examining the case of natural language texts. Such an example has been widely studied in the literature because of its importance for speech and handwriting recognition systems (see Chapter 12), but the considerations made for the language extend to many other cases.

The consequences of the sparseness can be observed in any collection of texts. Here we use the Wall Street Journal (WSJ) Corpus (year 1989), one of the main benchmarks used in the information retrieval literature (Table 10.1 reports the main characteristics of the Corpus). The size of the lexicon (the list of unique words appearing in the collection) is  $M = 72560$ , then the number of events  $(h, w)$  an  $N$ -gram has to take into account is  $M^N$ . For  $N = 2$ , the number of events is of the order of  $10^{12}$  while only  $\sim 5 \times 10^6$  events are available in the corpus. The number of unseen events is then six orders of magnitude higher than the number of seen events. This is the first consequence of the sparseness problem and it is strictly related to the dictionary size. In practice, only for small dictionaries it is possible to collect enough data to represent a higher fraction of the possible events. In the case of the language, many unseen events are not possible from a grammatical point of view and should not be taken into account. However, any attempt to identify and exclude such events



**Table 10.1.** Wall Street Journal Corpus. The table reports the main characteristics of the Wall Street Journal Corpus.

| Doc. Num. | Words Num | Avg. Length | Dict. Size |
|-----------|-----------|-------------|------------|
| 12380     | 5508825   | 445.0       | 72560      |

required heavy manual work without leading to significant improvements, then all possible events are typically included in the models.

The second problem is the reliability of the estimates obtained by maximizing the likelihood. Consider the number  $n_r$  of symbols appearing  $r$  times in a given data set. In the case of the texts, each word  $w_i$  is a symbol, and  $n_r$  is the number of words appearing exactly  $r$  times. Consider now the ranking in ascending order of the  $r$  values represented in the set:  $r_1$  is the smallest represented  $r$ ,  $r_2$  is the smallest  $r$  value with the exception of  $r_1$ , and so on. The so-called *Zipf law* [41] shows the relationship between  $n_r$  and the position  $pos(r)$  of  $r$  in the ranking:

$$n_r \simeq \frac{1}{pos(r)}. \quad (10.87)$$

The above relationship is observed experimentally in many natural phenomena where events can be ranked following the number of times they occur. In simple terms, the Zipf Law says that the number of events appearing  $r_1$  times is twice the number of events appearing  $r_2$  times, and so on. Figure 10.8 shows  $n_r$  as a function of  $pos(r)$  for the WSJ corpus, the singletons account for less than 1% of the word mass, but they account for one third of the dictionary, then for one third of the events ( $h, w$ ). In other words, around 33% of the events at disposition for the training are represented only once. It is difficult to identify the minimum number of times an event should be represented to enable reliable estimates, but we can still observe that roughly two thirds of the events are represented no more than five times (see Figure 10.8). For this reason, it is necessary to *smooth* the ML estimations that give too much importance to small differences of  $r$  that are probably due to random fluctuations rather than to actual differences in the frequency of the events. The curve in Figure 10.8 has been obtained using the WSJ corpus, but similar results are obtained using any other text collection or, more in general, any other collection including events produced by natural and technological phenomena.

It is important to notice that, unlike other cases, the lack of events cannot be addressed by simply increasing the size of the training set. In fact, the number of words in the lexicon is related to the size of the corpus (expressed in number of words) through an increasing monotone relationship known as *Heaps law* [21]:

$$M \simeq kL \quad (10.88)$$

where  $k$  is a constant and  $L$  is the number of words of the corpus (the corresponding curve for the WSJ corpus is shown in Figure 10.8). In other words, the sparseness is an inherent property of the text collections and does not depend on a simple lack of training data.

The next subsections present some of the main techniques addressing the problems of smoothing and unseen events probability estimation.

## 10.9 Discounting and Smoothing Methods for $N$ -gram Models\*\*

The previous sections show that the estimation of the  $N$ -gram models parameters is affected by the sparseness problem, i.e. most of the events appearing in the training set have a frequency too low to enable reliable estimations and many events that should be taken into account do not appear in the training set. The methods used to address such problems are referred to as *smoothing* or *discounting* techniques. In both cases, part of the probability of events observed in the training set is moved to unseen events. This has the twofold effect of providing a probability estimate for unseen events and of smoothing the probability estimate of seen events, i.e. of reducing the estimate differences due to small changes in  $N(h, w)$  likely to be caused by random effects.

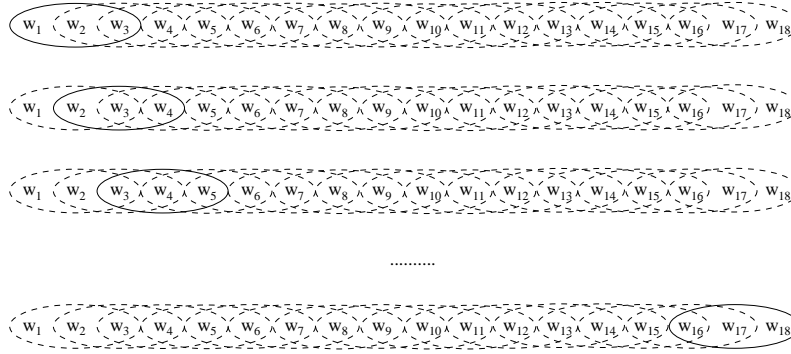
The next subsections present the so-called Turing Good counts [19] and the Katz's estimates [26], the most widely applied techniques addressing the above problems. Other techniques are available and extensive surveys can be found in [14][31]. The main idea behind such approaches is that the ML estimates can be modified as follows:

$$p(h, w) = \begin{cases} \frac{N(h, w)}{M^N} & N(h, w) = R \\ (1 - \lambda_{N(h, w)}) \frac{N(h, w)}{M^N} & 0 < N(h, w) < R \\ \frac{1}{n_0} \sum_{(h', w'): 0 < N(h', w') < R} \lambda_{N(h', w')} \frac{N(h', w')}{M^N} & N(h, w) = 0 \end{cases} \quad (10.89)$$

where  $n_r$  is the number of events  $(h, w)$  appearing  $r$  times in the training set,  $R = \max_{(h, w)} N(h, w)$  and  $\lambda_{N(h, w)}$  is the *discounting factor* for events represented  $N(h, w)$  in the training set. The problem is then to find the correct  $\lambda_r$  factors for events appearing  $r$  times.

### 10.9.1 The Leaving-One-Out Method

Consider a set of data that must be used to estimate the parameters of a model. A realistic measure of the model effectiveness can be obtained only if the test is performed over data separated and independent with respect to the data used for the training. In general, such a condition is respected by splitting the data into *training* and *test set* (see Chapter 4), but this is not always possible when there are few data at disposition. In this last case, it is



**Fig. 10.9.** The leave-one-out approach. The figure shows that a different event  $(h, w)$  is left out each time and used as test set.

more convenient to apply the *Leaving-one-out method* (LOO), i.e. a technique which uses alternatively each sample as a test set (see Figure 10.9).

The LOO can be used also as a basis for the discounting methods because it can simulate the effect of an event  $(h, w)$  not being observed. The consequence of holding out an event  $(h, w)$  is that the number of times  $N(h, w)$  such an event appears in the training set is decreased by one. This means that the discounting factor to be used for the event  $(h, w)$  is  $\lambda_{N(h,w)-1}$  rather than  $\lambda_{N(h,w)}$ . The loglikelihood of a model estimated using the LOO method is then as follows:

$$\begin{aligned}
 \log p(W|\{\lambda_r\}) &= \sum_{(h',w')} N(h', w') \log p(w'|h') \\
 &= \sum_{(h',w'):N(h',w')=1} \log p(w'|h') \\
 &\quad + \sum_{(h',w'):N(h',w')>1} N(h', w') \log p(w'|h') \\
 &= \sum_{(h',w'):N(h',w')=1} \log \left[ \frac{1}{n_0} \sum_{r=1}^{R-1} \lambda_r \frac{rn_r}{M^{N-1}} \right] \\
 &\quad + \sum_{(h',w'):N(h',w')>1} \log \left[ (1 - \lambda_{N(h,w)-1}) \frac{N(h,w)-1}{M^{N-1}} \right].
 \end{aligned} \tag{10.90}$$

Since the goal is the estimation of the discount factors, the only addends of interest are those containing the  $\lambda$  parameters:

$$\begin{aligned}
 \log \tilde{p}(W|\{\lambda_r\}) &= \sum_{(h',w'):N(h',w')=1} \log \left( \sum_{r=1}^{R-1} \lambda_r rn_r \right) \\
 &\quad + \sum_{(h',w'):N(h',w')>1} N(h, w) \log (1 - \lambda_{N(h,w)-1}) \\
 &= \sum_{(h',w'):N(h',w')=1} \log \left( \sum_{r=1}^{R-1} \lambda_r rn_r \right) \\
 &\quad + \sum_{r=2}^R \sum_{(h',w'):N(h',w')=r} r \log(1 - \lambda_{r-1}) \\
 &= n_1 \log \left( \sum_{r=1}^{R-1} \lambda_r rn_r \right) + \sum_{r=2}^R rn_r \log(1 - \lambda_{r-1}) \\
 &= n_1 \log \left( \sum_{r=1}^{R-1} \lambda_r rn_r \right) + \sum_{r=1}^{R-1} (r + 1)n_{r+1} \log(1 - \lambda_r).
 \end{aligned} \tag{10.91}$$

The last expression can be derived with respect to  $\lambda_r$  and set equal to zero in order to find the discounting factor estimates maximizing the likelihood:

$$\frac{\partial \tilde{p}(W|\{\lambda_r\})}{\partial \lambda_r} = n_1 \frac{rn_r}{\sum_{s=1}^{R-1} \lambda_s sn_s} - \frac{(r+1)n_{r+1}}{1-\lambda_r}. \quad (10.92)$$

The solution of such an equation system is:

$$\lambda_r = 1 - \left( \frac{\sum_{s=1}^{R-1} sn_s}{\sum_{s=1}^R sn_s} \right) \frac{(r+1)n_{r+1}}{rn_r} = 1 - \left( 1 - \frac{Rn_R}{M^{N-1}} \right) \frac{(r+1)n_{r+1}}{rn_r}. \quad (10.93)$$

By plugging the above expression into Equation (10.89), the  $p(h, w)$  estimates for  $0 < N(h, w) < R$  become:

$$p(h, w) = \left( 1 - \frac{Rn_R}{M^N} \right) \frac{(N(h, w) + 1)n_{N(h, w)+1}}{M^N n_{N(h, w)}}, \quad (10.94)$$

and the probability mass of the unseen events is:

$$\sum_{(h', w'): N(H', w')=0} \left( 1 - \frac{Rn_R}{M^T} \right) \frac{n_1}{M^T}. \quad (10.95)$$

The last expression shows how important it is the role of the *singletons* in estimating the probability mass of the unseen events. The reason is that an event appearing once should not have a probability much higher than an unseen event. In fact the simple presence or absence of  $(h, w)$  can be due to random fluctuations.

### 10.9.2 The Turing Good Estimates

Consider Equation (10.95), in general  $Rn_R/M^N \ll 1$  and the probabilities  $p(h, w)$  can be approximated as follows:

$$p(h, w) \simeq \frac{1}{M^N} \frac{[N(h, w) + 1]n_{N(h, w)+1}}{n_{N(h, w)}}. \quad (10.96)$$

This corresponds to the so-called *Turing Good estimates* which can be interpreted as a relative frequency count where the original count  $r = N(h, w)$  is replaced with a modified value  $r^*$  obtained through a discounting procedure:

$$r^* = \frac{(r+1)n_{r+1}}{n_r}, \quad (10.97)$$

where  $r$  is often referred to as *Turing Good count*. The same approximation holds for the discounting factors  $\lambda_r$ :

$$\lambda_r = 1 - \frac{(r+1)n_{r+1}}{rn_r} \quad (10.98)$$

and for the estimated probability mass of the unseen events:

$$\sum_{(h, w): N(h, w)=0} p(h, w) \simeq \frac{n_1}{M^N} \quad (10.99)$$

which shows once again the important role played by the singletons in estimating the probability of unseen events.

### 10.9.3 Katz's Discounting Model

The Turing Good estimates of Equation (10.97) are used as a starting point in another widely applied approach proposed in [26] and known as *Katz's discounting method*. The probability of an event  $(h, w)$  can be estimated using both maximum likelihood and Turing Good discounts, the difference between the corresponding values is:

$$p_{ML}(h, w) - p_T(h, w) = \frac{N(h, w)}{M^N} - \frac{N^*(h, w)}{M^N} = \delta_{N(h, w)}, \quad (10.100)$$

where  $N^*(h, w)$  is the Turing Good count:

$$N^*(h, w) = [N(h, w) + 1] \frac{n_{N(h, w)+1}}{n_{N(h, w)}}. \quad (10.101)$$

If the difference  $\delta_{N(h, w)}$  is summed over all events represented in the training corpus, the result is:

$$\sum_{(h, w): N(h, w) > 0} \delta_{N(h, w)} = \sum_{r > 0} n_r (1 - d_r) \frac{r}{M^N} = \frac{n_1}{M^N} \quad (10.102)$$

where  $d_r = r^*/r$ . In other words, the sum over the differences corresponds to the probability of the unseen events (see Equation (10.99)) and the single term  $\delta_{N(h, w)}$  can be thought of as the contribution given by the event  $(h, w)$  to the unseen events probability mass. Such an interpretation can be extended to the case where we consider conditional probabilities  $p(w|h)$  rather than joint probabilities  $p(h, w)$ , the  $\delta_{N(h, w)}$  changes as follows:

$$\delta_{N(h, w)}^{cond} = (1 - d_{N(h, w)}) \frac{N(h, w)}{N(h)}. \quad (10.103)$$

At this point, the estimates of  $p(w|h)$  can be obtained by induction. If  $h = h_1^n$ , we can define  $h_- = h_2^n$ , then  $p(w|h_-)$  corresponds to a model of order  $N - 1$ . Since we are defining the  $p(w|h)$  by induction, we can consider the  $p(w|h_-)$  as given. If  $N(h) > 0$ , the probability  $p(w|h)$  can be estimated as:

$$\tilde{p}(w|h) = \frac{N^*(h, w)}{N(h)} = d_{N(h, w)} \frac{N(h, w)}{N(h)}, \quad (10.104)$$

and this enables one to define a function  $\beta(h)$  that accounts for the probability of events  $(h, w')$  not observed in the training set:

$$\beta(h) = \sum_{w': N(h, w') > 0} \delta_{N(h, w')}^{cond} = 1 - \sum_{w': N(h, w') > 0} \tilde{p}(w|h). \quad (10.105)$$

The probability mass  $\beta(h)$  can be distributed across all symbols  $w'$  such that  $N(h, w') = 0$  by using the estimate  $p(w'|h_-)$ :

$$p(w'|h) = \frac{\beta(h)}{\sum_{w:N(h,w)=0} p(w|h_-)} p(w'|h_-) = \alpha(h)p(w'|h_-). \quad (10.106)$$

In other words, the conditional probability of the unseen event  $(h, w')$  is obtained as a product between the lower order probability  $p(w'|h_-)$  and the normalizing constant  $\alpha(h)$ .

The above applies to the case where  $N(h) > 0$ . If  $N(h) = 0$ , then  $\tilde{p}(w|h) = 0$  and  $\beta(h) = 1$ . In other words, when an event of a certain order  $(h, w)$  is unseen, its probability is estimated using  $p(w|h_-)$ , i.e. the probability of the immediately lower order event  $(h_-, w)$ . If the event  $(h_-, w)$  is unseen, the order is further lowered until the event is observed. The two cases can be summarized in a single expression:

$$p(w|h) = \tilde{p}(w|h) + I(\tilde{p}(w|h))\alpha(h)p(w|h_-), \quad (10.107)$$

where  $I(x)$  is defined as follows:

$$I(x) = \begin{cases} 1 & x = 0 \\ 0 & x > 0. \end{cases} \quad (10.108)$$

The use of lower order estimates to address the lack of events in the training data is often referred to as *backing off*.

If the discounting is applied only to events appearing less than  $k+1$  times,  $d_r = 1$  for  $r > k$  and it holds the following:

$$\sum_{r=1}^k n_r(1 - d_r) \frac{r}{M^T} = \frac{n_1}{M^T} \quad (10.109)$$

which leads to:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}. \quad (10.110)$$

since  $d_r = r'/r$ , i.e.  $d_r$  is the ratio between the counts after discount and the actual counts, the above equation means that:

$$r' = \frac{r^* - \frac{(k+1)n_{k+1}}{n_1}r}{1 - \frac{(k+1)n_{k+1}}{n_1}} \quad (10.111)$$

and the Katz's discounting can be interpreted as a smoothing operation performed over the Turing Good counts  $r^*$ .

## 10.10 Building a Language Model with $N$ -grams

This section provides a quick tutorial on the use of the *SLM-CMU toolkit* [15] a free software package aimed at creating statistical language models (SLM)

based on  $N$ -grams.<sup>2</sup> Although created explicitly for SLM, the toolkit can be easily applied for any other kind of problems involving sequences of symbols belonging to a finite alphabet.

The package includes several functions (listed and described in [15] and in the documentation of the package) that bring from the raw input to the  $N$ -gram model. As an example, we consider the WSJ corpus described in Section 10.8.4, the raw input is an ASCII file containing the whole corpus without any other kind of information (e.g. tags or document delimiters). The models are built through a sequence of steps:

1. *Extraction of the word frequencies.* The command `textwfreq` takes as input the raw text and gives as output a file containing all unique words  $w$  appearing in the corpus together with their frequencies  $N(w)$ .
2. *Extraction of the dictionary.* The command `wfreq2vocab` takes as input the counts  $N(w)$  produced at the previous step and gives as output a dictionary. The options enable one to include all the words appearing more than a certain number  $k$  of times or to include the  $k'$  most frequent words.
3. *Extraction of the  $N$ -gram counts.* The command `text2idngram` takes as input the raw text and the dictionary produced at the previous step and gives as output a file containing all  $N$ -grams  $(h, w)$  with respective counts  $N(h, w)$ . The options enable one to select the order  $N$ .
4. *Extraction of the  $N$ -gram model.* The command `idngram2lm` takes as input the file of the  $N$ -gram counts produced at the previous step and the dictionary, and gives as output the language model. The options enable one to select discounting strategy, output format, etc.

The above description includes only the basic options, but the package offers more possibilities and parameters to optimize the models. Moreover, some functions (not described here) provide some statistics about the content of the corpus.

## Problems

10.1. Consider the HMM  $\lambda = \{\boldsymbol{\pi}, A, B\}$  where  $\boldsymbol{\pi} = (1, 0, 0)$ ,

$$\boldsymbol{\pi} = (1, 0, 0); A = \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.3 & 0.1 & 0.6 \end{pmatrix}; B = \begin{pmatrix} 0.5 & 0.5 & 0.0 \\ 0.3 & 0.3 & 0.4 \\ 0.5 & 0.3 & 0.2 \end{pmatrix} \quad (10.112)$$

What the likelihood of the observation sequence  $\mathbf{x}_1^3 = (c_2, c_2, c_1)$ ? If the transition matrix is modified as follows:

<sup>2</sup> The package can be downloaded for free from the site of the University of Cambridge: [svr-www.eng.cam.ac.uk/~prc14/toolkit.html](http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html) and it can be easily installed on several platforms. At the moment of writing this book the site is still active although the authors of the code do not follow directly the development any more.

$$A = \begin{pmatrix} 0.4 & 0.2 & 0.4 \\ 0.3 & 0.6 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{pmatrix}, \quad (10.113)$$

what is the likelihood of the same sequence  $\mathbf{x}_1^3$ ?

**10.2.** Demonstrate that the product  $\alpha_t(i)\beta_t(i)$  can be used to estimate the probability of passing through state  $v_i$  at time  $t$  using the following equation (see Section 10.5):

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{p(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (10.114)$$

**10.3.** Demonstrate that the estimates of the GM coefficients in a continuous density HMM can be obtained using the  $\gamma$  variables as follows:

$$c_{kn}^{(i)} = \frac{\sum_{t=1}^T \gamma_{tl}(k)}{\sum_{t=1}^T \gamma_t(k)} \quad (10.115)$$

See Equations (10.64) and (10.65) for the meaning of the symbols.

**10.4.** Demonstrate that the estimates of the initial state probabilities of an HMM correspond to the  $\gamma$  variables defined in Equation (10.38):

$$\pi_k^{(i)} = \gamma_1(k) \quad (10.116)$$

**10.5.** Demonstrate that the following variable:

$$\xi_t(i, j) = \frac{\sum_{t=2}^T \sum_{n=1}^N p(O, s_{t-1} = v_m, s_t = v_n | \lambda^{(i-1)})}{p(O | \lambda^{(i-1)})} \quad (10.117)$$

can be computed using  $\alpha_t(i)$  and  $\beta_t(i)$ :

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(\mathbf{x}_{t+1})\beta_{t+1}(j)}{p(O|\lambda)}. \quad (10.118)$$

**10.6.** Demonstrate that the derivative of the following expression:

$$= \sum_{i=1}^N \sum_{l=1}^G \sum_{t=1}^T \log(w_{il} \mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{il}^{(i)}, \Sigma_{il}^{(i)})) p(O, s_t = v_i, m_{s_t, t} = l | \lambda^{(i-1)}) \quad (10.119)$$

with respect to  $\boldsymbol{\mu}_{kn}$  is:

$$\sum_{t=1}^T \Sigma_{kn}(\mathbf{x}_t - \boldsymbol{\mu}_{kn}) p(O, s_t = v_k, m_{s_t, t} = n | \lambda^{(i-1)}) \quad (10.120)$$



**10.7.** Consider the toolkit described in Section 10.10. Extract the counts  $N(w)$  from a corpus of sequences and plot  $n_r$  as a function of  $pos(r)$  (see Section 10.8.4 for the meaning of symbols). Is the plot different from Figure 10.8? If yes provide some explanations.

**10.8.** Consider the toolkit described in Section 10.10. Extract an  $N$ -gram model from a corpus of sequences using different discounting strategies and measure the corresponding perplexities over a test set different from the data used for the training. Identify the discounting strategy leading to the best results.



---

## References

1. P. Baldi and S. Brunak. *Bioinformatics: The Machine Learning Approach*. MIT Press, 2001.
2. L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8, 1972.
3. L.E. Baum and J. Eagon. An inequality with applications to statistical prediction for functions of Markov processes and to a model of ecology. *Bulletin of the American Mathematical Society*, 73:360–363, 1967.
4. L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.
5. R. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.
6. S. Bengio. An asynchronous hidden Markov model for audio-visual speech recognition. In *Advances in Neural Information Processing Systems*, pages 1237–1244, 2003.
7. Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.
8. Y. Bengio and P. Frasconi. An inout/output HMM architecture. In *Advances in Neural Information Processing Systems*, pages 427–434, 1995.
9. J.A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden markov models. Technical Report TR-97-021, International Computer Science Institute (ICSI), Berkeley, 1998.
10. R. Bjar and S. Hamori. *Hidden Markov Models: Applications to Financial Economics*. Springer-Verlag, 2004.
11. H. Bourlard and S. Bengio. Hidden Markov models and other finite state automata for sequence processing. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. 2002.
12. H. Bourlard, Y. Konig, and N. Morgan. A training algorithm for statistical sequence recognition with applications to transition-based speech recognition. *IEEE Signal Processing Letters*, 3(7):203–205, 1996.
13. H. Bourlard and N. Morgan. *Connectionist Speech Recognition - A Hybrid Approach*. Kluwer Academic Publishers, 1993.

14. S. Chen and R. Rosenfeld. A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50, 2000.
15. P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge Toolkit. In *Proceedings of Eurospeech*, pages 2707–2710, 1997.
16. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
17. R.J. Elliott, L. Aggoun, and J.B. Moore. *Hidden Markov Models: Estimation and Control*. Springer-Verlag, 1997.
18. S. Godfeld and R. Quandt. A markov model for switching regressions. *Journal of Econometrics*, 1:3–16, 1973.
19. I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.
20. J. Hamilton. A new approach of the economic analysis of non-stationary time series and the business cycle. *Econometrica*, 57:357–384, 1989.
21. H.S. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, 1978.
22. J. Hennebert, C. Ris, H. Boudlard, S. Renals, and N. Morgan. Estimation of global posteriors and forward-backward training of hybrid HMM-ANN systems. In *Proceedings of Eurospeech*, pages 1951–1954, 1997.
23. J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Language and Computations*. Addison Wesley, 2000.
24. F. Jelinek. *Statistical Aspects of Speech Recognition*. MIT Press, 1997.
25. R. Kalman and R. Bucy. New results in linear filtering and prediction. *Journal of Basic Engineering*, 83D:95–108, 1961.
26. S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.
27. D. Klakow and J. Peters. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1):19–28, 2002.
28. T. Koski. *Hidden Markov Models for Bioinformatics*. Springer-Verlag, 2002.
29. A. Markov. An example of statistical investigation in the text of Eugene Onyegin illustrating coupling of test in chains. *Proceedings of the Academy of Sciences of St. Petersburg*, 1913.
30. G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley, 1997.
31. H. Ney, S. Martin, and F. Wessel. Statistical language modeling. In S. Young and G. Bloothoof, editors, *Corpus Based Methods in Language and Speech Processing*, pages 174–207. Kluwer Academic Publishers, 1997.
32. L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.F. Lee, editors, *Readings in Speech Recognition*, pages 267–296. 1989.
33. D. Ron, Y. Singer, and N. Tishby. The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.
34. D. Ron, Y. Singer, and N. Tishby. Learning with probabilistic automata with variable memory length. In *Proceedings of ACM Conference on Computational Learning Theory*, pages 35–46, 1997.
35. R. Rosenfeld. Two decades of Statistical Language Modeling: where do we go from here? *Proceedings of IEEE*, 88(8):1270–1278, 2000.

36. R. Shumway and D. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 4(3):253–264, 1982.
37. E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R.C. Carrasco. Probabilistic finite state machines - Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005.
38. E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R.C. Carrasco. Probabilistic finite state machines - Part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005.
39. A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.
40. L. Xu and M.J. Jordan. On convergence properties of the EM algorithm for Gaussian Mixtures. *Neural Computation*, 8:129–151, 1996.
41. G. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.



## Feature Extraction Methods and Manifold Learning Methods

### What the reader should know to understand this chapter

- Notions of calculus.
- Chapter 4.

### What the reader should know after reading this chapter

- Curse of dimensionality.
- Intrinsic dimensionality.
- Principal component analysis.
- Independent component analysis.
- Multidimensional scaling algorithms.
- Manifold learning algorithms (Isomap, LLE, Laplacian eigenmaps).

### 11.1 Introduction

In the previous chapters we presented several learning algorithms for classification and regression tasks. In many applicative problems data cannot be straightaway used to feed learning algorithms; they first need to have undergone a preliminary *preprocessing*. To illustrate this concept, we consider the following example. Suppose we want to build an automatic handwriting character recognizer, that is a system able to associate to a given bitmap the correct alphabet letter or digit. We assume that the data have the same sizes, that the data are bitmaps of  $n \times m$  pixels; for the sake of simplicity we assume  $n = m = 2^8$ . Therefore the number of possible configurations is  $2^8 \times 2^8 = 2^{16}$ . This consideration implies that a learning machine straightly fed by character bitmaps will perform poorly since a representative training set can not be built. A common approach for overcoming this problem consists in representing each bitmap by a vector of  $d$  (with  $d \ll nm$ ) measures computed on the bitmap, called *features*, and then feeding the learning machine with

the *feature vector*. The feature vector has the aim of representing in a concise way the distinctive characteristics of each letter. The more features represent the distinctive characteristics of each single character the higher is the performance of the learning machine. In machine learning, the preprocessing stage that converts the data into feature vectors is called *feature extraction*. One of the main aims of the feature extraction is to obtain the most representative feature vector using a number as small as possible of features. The use of more features than strictly necessary leads to several problems. A problem is the space needed to store the data. As the amount of available information increases, the compression for storage purposes becomes even more important. The speed of learning machines using the data depends on the dimension of the vectors, so a reduction of the dimension can result in reduced computational time. The most important problem is the sparsity of data when the dimensionality of the features is high. The sparsity of data implies that it is usually hard to make learning machines with good performances when the dimensionality of input data (that is, the feature dimensionality), is high. This phenomenon, discovered by Bellman, is called the *curse of dimensionality* [7].

The reasons presented above indicate that an important goal in the feature extraction consists in reducing the dimensionality of the input data. Firstly, it can be performed selecting the features that have high discriminant power. This activity cannot be always performed. For instance, as in applications of voice information retrieval, we can have a feature vector formed by hundreds of features. Although the discriminative power of each feature is small, the contribution of each feature cannot be omitted; otherwise the learning machine performance degrades. In this case an approach consists of projecting the original data by means of a nonlinear mapping onto a subspace of dimensionality lower than the original one. These techniques are justified by the observation that even if data are embedded in  $\mathbb{R}^d$  this does not necessarily imply that its actual dimension is  $d$ . Figure 11.2 shows a set  $\Omega$  of data points lying on a semicircle. Therefore the dimensionality of  $\Omega$  is 1, although the points are embedded in  $\mathbb{R}^2$ . Intuitively, the dimensionality (or *intrinsic dimensionality*) [30] of a data set is the minimum number of free variables needed to represent the data without information loss. Several feature extraction methods can have remarkable advantages with the knowledge of the dimensionality of the original data.

The aim of this chapter is to introduce the main methods of feature extraction, paying special attention to dimensionality reduction methods. The chapter is organized as follows: In Section 11.2 the curse of dimensionality is described in the framework of the function approximation theory. Section 11.3 presents the concept of data dimensionality and describes some algorithms to estimate it. Sections 11.4 and 11.5 review Principal and Independent Component Analysis. Some methods of Multidimensional Scaling are presented in Section 11.6. Section 11.7 introduces the problem of manifold learning and describes the main manifold learning algorithms. Finally, some conclusions are drawn in Section 11.8.



## 11.2 The Curse of Dimensionality\*

In this section we will discuss the curse of dimensionality in the framework of *function approximation theory*. The reading of this section can be omitted by practitioners and readers not interested to this topic. In Chapter 7 we saw that the training error  $E_{train}$  and the test error  $E_{test}$  of the learning machine are connected by means of the following inequality:

$$E_{test} \leq E_{train} + E_{est}$$

where  $E_{est}$  is *estimation error*.

We want to estimate the training error of the learning machine using the function approximation theory. Following the approach proposed in [33], we consider a normed space of function  $\Phi$  and a subset of  $\Phi$ ,  $F$ . The goal of the function approximation theory is to approximate a function  $\phi$  of  $\Phi$  ( $\phi \in \Phi$ ) by means of another function  $f$  ( $f \in F$ ) that belongs to  $F$ . This can be formalized as looking for an element in  $F$  whose distance from  $\phi$  is minimal. If we define the *distance of  $\phi$  from  $F$*   $\delta(\phi, F)$  as follows:

$$\delta(\phi, F) = \inf_{f \in F} \|\phi - f\| \quad (11.1)$$

the aim of approximation theory is to study  $\delta(\phi, F)$  for different subsets  $F$  and function to approximate  $\phi$ . In the linear theory of the approximation [60]  $F$  is a linear  $k$ -dimensional subspace (e.g. the polynomials of a given degrees or splines with fixed knots). Whereas in the nonlinear approximation theory  $F$  is a  $k$ -dimensional nonlinear *manifold* [21]. According to the approximation theory, there is usually a family of manifolds  $\{F_k\}_{k=1}^{\infty}$  such that  $\cup_k F_k$  is dense in  $\Phi$  and

$$F_1 \subset F_2 \subset \dots \subset F_n \subset \dots$$

hence  $\delta(\phi, F)$  is a monotone decreasing function that converges to zero. Therefore if we can get an  $F_k$  arbitrarily close to  $\phi$  picking a  $k$  adequately large. An interesting parameter is the convergence rate of  $\delta(\phi, F)$  to zero. For the linear approximation theory it can be shown [60] that  $\delta(\phi, F)$  cannot exceed the following bound:

$$\delta(\phi, F) = O(d^n \frac{s+\alpha}{d}) \quad (11.2)$$

where  $n$  is the the number of parameters (e.g.  $k$ ),  $d$  is the dimension of the input space,  $\alpha$  is a positive constant,  $s$  the smoothness index of  $\phi$ , that can be assumed equal to the number of bounded derivatives of the function.

We can observe that due to presence of  $d$  in denominator of the exponent, the rate of convergence exponentially decreases when the dimension increases. Therefore Equation (11.2) is the theoretical justification of the *curse of dimensionality*. Similar results have been obtained in the nonlinear approximation theory [21]. For sake of completeness, we quote that there are results [4][9][34][45] for particular function spaces with rates of the convergence  $O(\frac{1}{\sqrt{n}})$ . Although these results seem to suggest that some application

schemes are not subjected to the curse of dimensionality in particular cases, their utility is quite limited. In most cases the function spaces  $\mathcal{F}$  for which the curse of dimensionality does not hold are so specific that it is not clear if they are adequately large to include functions that are usually encountered in typical machine learning applications. Therefore we can assume that bound (11.2) is correct for almost the totality of the machine learning problems.

### 11.3 Data Dimensionality

In this section we introduce the concept of *data dimensionality* whose knowledge can be very useful to develop reliable feature extraction methods. Machine learning usually deals with data represented as vectors of dimension  $d$ . The data is then embedded in  $\mathbb{R}^d$ , but this does not necessarily imply that its actual dimension is  $d$ . The dimensionality of a data set is the minimum number of free variables needed to represent the data without information loss. In more general terms, following Fukunaga [30], a data set  $\Omega \subset \mathbb{R}^d$  is said to have *intrinsic dimensionality (ID)* equal to  $M$  if its elements lie entirely within an  $M$ -dimensional subspace of  $\mathbb{R}^d$  (where  $M < d$ ).

Following the classification proposed in [44], there are two approaches for estimating ID. In the first one (*local*) ID is estimated using the information contained in sample neighborhoods, avoiding the projection of the data onto a lower-dimensional space. In the second approach (*global*), the data set is unfolded in the  $d$ -dimensional space. Unlike local approaches that use only the information contained in the neighborhood of each data sample, global approaches make use of the whole data set.

#### 11.3.1 Local Methods

Local (or *topological*) methods try to estimate the topological dimension of the data manifold. The definition of topological dimension was given by Brouwer [37] in 1913. Topological dimension is the basis dimension of the local linear approximation of the hypersurface on which the data resides, i.e. the tangent space. For example, if the data set lies on an  $m$ -dimensional submanifold, then it has an  $m$ -dimensional tangent space at every point in the set. For instance, a sphere has a two-dimensional tangent space at every point and may be viewed as a two-dimensional manifold. Since the ID of the sphere is three, the topological dimension represents a lower bound of the ID. If the data does not lie on a manifold, the definition of topological dimension does not directly apply. Sometimes the topological dimension is also referred to simply as the *local dimension*. This is the reason why the methods that estimate the topological dimension are called local. The basic algorithm to estimate the topological dimension was proposed by Fukunaga and Olsen [32]. Alternative approaches to the Fukunaga-Olsen's algorithm have been proposed to estimate

locally the ID. Among them the methods [68][87] based on *near neighbor algorithm* [86] and the methods [10] based on *topological representing networks* (TRN) [63] are the most popular.

### Fukunaga-Olsen's Algorithm

*Fukunaga-Olsen's algorithm* is based on the observation that, for vectors embedded in a linear subspace, the dimension is equal to the number of nonzero eigenvalues of the covariance matrix. Besides, Fukunaga and Olsen assume that the intrinsic dimensionality of a data set can be computed by dividing the data set in small regions (*Voronoi tessellation* of data space). Voronoi tessellation can be performed by means of a clustering algorithm, e.g. LBG [59]. In each region (*Voronoi set*) the surface in which the vectors lie is approximately linear and the eigenvalues of the local covariance matrix are computed. Eigenvalues are normalized by dividing them by the largest eigenvalue. The intrinsic dimensionality is defined as the number of normalized eigenvalues that are larger than a threshold  $T$ . Although Fukunaga and Olsen proposed for  $T$ , on the basis of heuristic motivations, values such as 0.05 and 0.01, it is not possible to fix a threshold value  $T$  good for every problem.

#### 11.3.2 Global Methods

Global methods try to estimate the ID of a data set, unfolding the whole data set in the  $d$ -dimensional space. Unlike local methods that use only the information contained in the neighborhood of each data sample, global methods make use of the whole data set.

Among global methods [11][18][58], we describe *Fractal-Based Methods* since they are easy to be implemented. Fractal-based techniques are global methods that have been successfully applied to estimate the attractor dimension of the underlying dynamic system generating time series [47]. Unless other global methods, they can provide as ID estimation a non-integer value. Since fractals are generally<sup>1</sup> characterized by a non-integer dimensionality, for instance the dimension of Cantor's set and Koch's curve [62] is, respectively,  $\frac{\ln 2}{\ln 3}$  and  $\frac{\ln 4}{\ln 3}$ , these methods are called *fractal*.

In nonlinear dynamics many definitions of *fractal* dimensions [23] have been proposed. The *box-counting* and the *correlation* dimension are the most popular. The first definition of dimension (*Hausdorff dimension*) [23][66] is due to Hausdorff [36]. The *Hausdorff dimension*  $D_H$  of a set  $\Omega$  is defined by introducing the quantity

$$\Gamma_H^d(r) = \inf_{s_i} \sum_i (r_i)^d \quad (11.3)$$

---

<sup>1</sup> Fractals have not always noninteger dimensionality. For instance, the dimension of *Peano's curve* is 2.

where the set  $\Omega$  is covered by cells  $s_i$  with variable diameter  $r_i$ , and all diameters satisfy  $r_i < r$ .

That is, we look for that collection of covering sets  $s_i$  with diameters less than or equal to  $r$  that minimizes the sum in (8), and we denote the minimized sum  $\Gamma_H^d(r)$ . The *d-dimensional Hausdorff measure* is then defined as

$$\Gamma_H^d = \lim_{r \rightarrow 0} \Gamma_H^d(r). \quad (11.4)$$

The *d-dimensional Hausdorff measure* generalizes the usual notion of the total length, area and volume of simple sets. Hausdorff proved that  $\Gamma_H^d$ , for every set  $\Omega$ , is  $+\infty$  if  $d$  is less than some critical value  $D_H$  and is 0 if  $d$  is greater than  $D_H$ . The critical value  $D_H$  is called the *Hausdorff dimension* of the set.

Since the Hausdorff dimension is not easy to evaluate, in practical application it is replaced by an upper bound that differs only in some constructed examples: the *box-counting dimension* (or *Kolmogorov capacity*) [66].

The box-counting dimension  $D_B$  of a set  $\Omega$  is defined as follows:

if  $\nu(r)$  is the number of the boxes of size  $r$  needed to cover  $\Omega$ , then  $D_B$  is

$$D_B = \lim_{r \rightarrow 0} \frac{\ln(\nu(r))}{\ln(\frac{1}{r})}. \quad (11.5)$$

It can be shown that if in the definition of Hausdorff dimension the cells have the same diameter  $r$ , Hausdorff dimension reduces to box-counting dimension. Although efficient algorithms [50] have been proposed, the box-counting dimension can be computed only for low-dimensional sets because the algorithmic complexity grows exponentially with the set dimensionality.

A good substitute for the box-counting dimension can be the *correlation dimension* [35]. Due to its computational simplicity, the correlation dimension is successfully used to estimate the dimension of attractors of dynamical systems.

The correlation dimension is defined as follows:

Let  $\Omega = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\ell$  be a set of points in  $\mathbb{R}^n$ . If the *correlation integral*  $C_m(r)$  is defined as:

$$C_m(r) = \lim_{N \rightarrow \infty} \frac{2}{\ell(\ell-1)} \sum_{i=1}^{\ell} \sum_{j=i+1}^{\ell} I(\|\mathbf{x}_j - \mathbf{x}_i\| \leq r) \quad (11.6)$$

where  $I$  is an *indicator function*,<sup>2</sup> then the *correlation dimension*  $D$  of  $\Omega$  is:

$$D = \lim_{r \rightarrow 0} \frac{\ln(C_m(r))}{\ln(r)}. \quad (11.7)$$

Correlation and box-counting dimensions are strictly related. It can be shown that both dimensions are special cases of the *generalized Renyi dimension*.

<sup>2</sup>  $I(\lambda)$  is 1 iff condition  $\lambda$  holds, 0 otherwise.

If the *generalized correlation integral*  $C_p$  is:

$$C_p(r) = \frac{1}{\ell(\ell-1)^{p-1}} \sum_{i=1}^{\ell} \left( \sum_{j \neq i}^{\ell} I(\|\mathbf{x}_j - \mathbf{x}_i\| \leq r) \right)^{p-1} \quad (11.8)$$

The generalized Renyi dimension  $D_p$  is defined in the following way:

$$D_p = \lim_{r \rightarrow 0} \frac{1}{p-1} \frac{\ln(C_p(r))}{\ln(r)} \quad (11.9)$$

It can be shown [35] that for  $p = 0$  and  $p = 2$   $D_p$  reduces, respectively, to the box-counting and the correlation dimension. Besides, it can be proved that correlation dimension is a lower bound of the box-counting dimension. Nevertheless, due to noise, the difference between the two dimensions is negligible in applications with real data.

### Methods of Estimation of Fractal Dimension

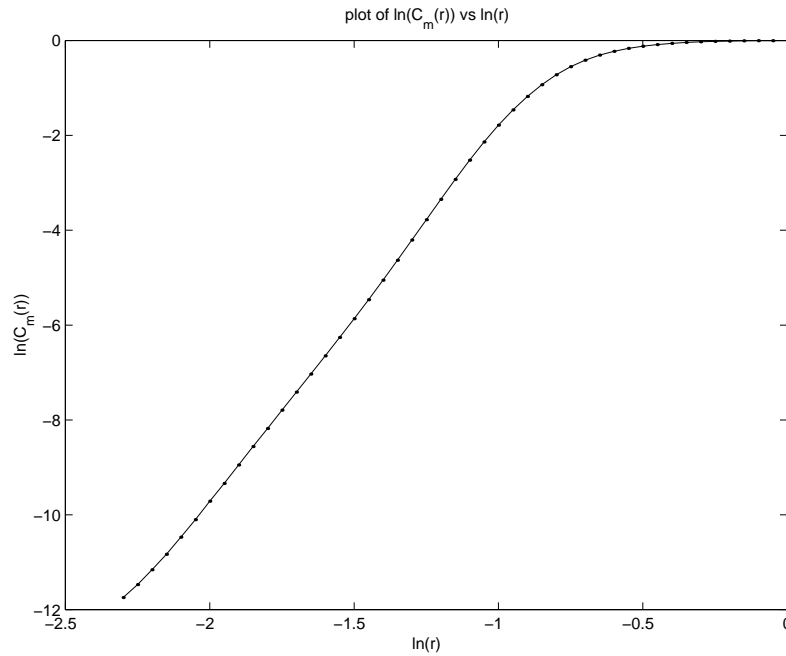
The most popular method to estimate box-counting and correlation dimension is the *log-log plot*. This method consists in plotting  $\ln(C_m(r))$  versus  $\ln(r)$ . The correlation dimension is the slope of the linear part of the curve (Figure 11.1). The method to estimate box-counting is analogous, but  $\ln(\nu(r))$  replaces  $\ln(C_m(r))$ . The methods to estimate correlation and box-counting dimension present some drawbacks. Though correlation and box-counting dimension are asymptotic results and hold only for  $r \rightarrow 0$ ;  $r$  cannot be too small since too few observations cannot allow to get reliable dimension estimates. In fact the noise has most influence at small distance. Therefore there is a trade-off between taking  $r$  small enough to avoid non-linear effects and taking  $r$  sufficiently large to reduce statistical errors due to lack of data. The use of least-squares method makes the dimension estimate not adequately robust towards the outliers.

Some methods [80][81] have been studied to obtain an optimal estimate for the correlation dimension. Takens [81] has proposed a method, based on *Fisher's method of maximum likelihood* [22] [31], that allows us to estimate the correlation dimension with a standard error. Takens' method is the following.

Let  $Q$  be the set  $Q = \{q_k \mid q_k < r\}$  where  $r_k$  is the the Euclidean distance between a generic couple of points of  $\Omega$  and  $r$  (*cut-off radius*) is a real positive number.

Using the maximum likelihood principle it can prove that the expectation value of the correlation dimension  $\langle D_c \rangle$  is:

$$\langle D_c \rangle = - \left( \frac{1}{|Q|} \sum_{k=1}^{|Q|} q_k \right)^{-1} \quad (11.10)$$



**Fig. 11.1.** Plot of  $\ln(C_m(r))$  vs  $\ln(r)$ .

where  $|Q|$  stands for the cardinality of  $Q$ .

Takens' method presents some drawbacks. It requires some heuristics to set the radius [84]. Besides, the method is optimal only if the correlation integral  $C_m(r)$  assumes the form  $C_m(r) = ar^D[1 + br^2 + o(r^2)]$  where  $a$  and  $b$  are constants. Otherwise Takens' estimator can perform poorly [83].

### Limitations of Fractal Methods

In addition to the drawbacks previously exposed, estimation methods based on fractal techniques have a fundamental limitation.

It has been proved [24][79] that in order to get an accurate estimate of the dimension  $D$ , the set cardinality  $\ell$  has to satisfy the following inequality:

$$D < 2 \log_{10} \ell. \quad (11.11)$$

The *Eckmann-Ruelle inequality* (11.11) shows that the number  $N$  of data points needed to accurately estimate the dimension of a  $D$ -dimensional set is at least  $10^{\frac{D}{2}}$ . Even for low-dimensional sets this leads to huge values of  $N$ .

In order to cope with this problem and to improve the reliability of the measure for low values of  $N$ , the *method of surrogate data* [85] has been proposed. The method of surrogate data is an application of a well-known

statistic technique called *bootstrap* [25]. Given a data set  $\Omega$ , the method of surrogate data consists of creating a new synthetic data set  $\Omega'$ , with greater cardinality, that has the same statistical properties of  $\Omega$ , namely the same mean, variance and Fourier Spectrum. Although the cardinality of  $\Omega'$  can be chosen arbitrarily, the method of surrogate data cannot be used when the dimensionality of the data set is high. As pointed out previously, a data set whose dimension is 18 requires at least, on the base of (11.11), a data set with  $10^9$  points. Therefore the method of surrogate data becomes computationally burdensome.

Finally, heuristic methods [12] have been proposed in order to estimate how fractal techniques underestimate the dimensionality of a data set when its cardinality is unadequate. These heuristic methods permit inferring the actual dimensionality of the data set. Since the methods are not theoretically well-grounded they have to be used with prudence.

## 11.4 Principal Component Analysis

In this section we introduce the most common algorithm for the reduction the data dimensionality, i.e the *principal component analysis (PCA)* or *Karhunen-Loeve transform*.

Let  $\Omega = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$  be a data set, formed by vectors  $\mathbf{x}_i \in \mathbb{R}^n$ , which has mean  $\langle \mathbf{x} \rangle$  and covariance matrix  $\Sigma$ . Then we introduce the eigenvalue equation

$$\Sigma U = U \Lambda \quad (11.12)$$

where  $U$  is a  $n \times n$  matrix, consisting of  $N$  eigenvectors as  $U = [\mathbf{u}_1, \dots, \mathbf{u}_n]$  and  $\Lambda$  is a diagonal matrix of eigenvalues as

$$\begin{bmatrix} \lambda_1 & 0 & \cdots \\ 0 & \ddots & 0 \\ 0 & \cdots & \lambda_n \end{bmatrix}$$

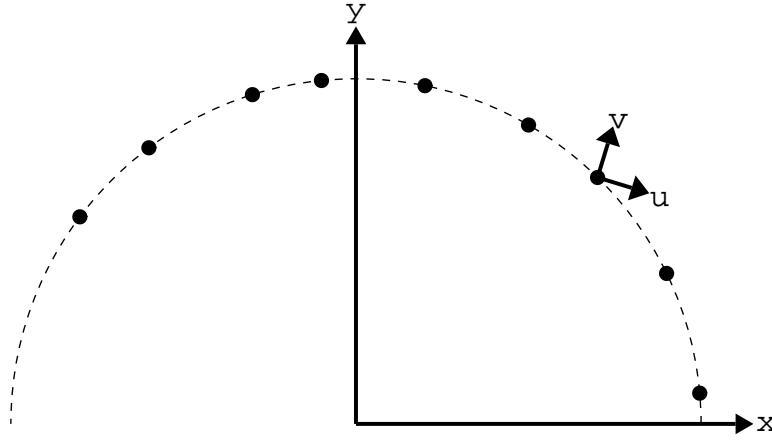
Each of  $\mathbf{u}_i$  component is called *principal component*. Since if  $i \neq j$  then  $\mathbf{u}_i \cdot \mathbf{u}_j = 0$  the principal components are *uncorrelated*.

We can define a new transformation of data that maps the data matrix  $X$  in a new matrix  $Y$ , given by:

$$Y = U^T X \quad (11.13)$$

It can be shown that PCA projects the data along the directions of maximal variance [1].

Principal component analysis is strictly connected [51] to a standard decomposition in numerical analysis, namely the *singular value decomposition (SVD)*.



**Fig. 11.2.**  $\Omega$  Data set. The data set is formed by points lying on the upper semi-circumference of equation  $x^2 + y^2 = 1$ . The ID of  $\Omega$  is 1. Nevertheless PCA yields *two* non-null eigenvalues. The principal components are indicated by  $u$  and  $v$ .

PCA can be used for estimating the intrinsic data dimensionality. ID is given by the number of non-null eigenvalues. PCA is a poor dimensionality estimator, since it tends to overestimate the ID [8]. As shown in Figure 11.2, a data set formed by points lying on a circumference for PCA has dimension 2 rather than 1.

Although PCA is a poor ID estimator, PCA is widely used for reducing the data dimensionality. Suppose to order the eigenvectors  $\mathbf{u}_1, \dots, \mathbf{u}_N$  on the basis of size of the respective eigenvalues  $\lambda_1, \dots, \lambda_p$ . In this way the eigenvector  $\mathbf{u}_1$  has the largest eigenvalue  $\lambda_1$  and in general to the  $k^{\text{th}}$  eigenvector  $\mathbf{u}_k$  corresponds the  $k^{\text{th}}$  largest eigenvalue  $\lambda_k$ . We pick the first  $k$  eigenvectors and we discard the remaining  $N - k$  eigenvectors. In this way we project our original data  $\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N$  onto a vector  $\mathbf{x}' = (x'_1, \dots, x'_k) \in \mathbb{R}^K$  in a  $K$ -dimensional space, where  $K < N$ . Discarding the last  $N - k$  eigenvectors we assume that the data information is contained in the first  $k$  components whereas the last  $N - k$  components contain noise.

Now we evaluate the information lost discarding last  $N - k$  eigenvectors [8]. Let  $\mathbf{x}'$  be the projection of  $\mathbf{x}$  considering all  $N$  principal components. Whereas let  $\mathbf{x}''$  be the projection considering the first  $K$  principal components. We have

$$\mathbf{x}' = \sum_{i=1}^N x'_i \mathbf{u}_i; \quad \mathbf{x}'' = \sum_{i=1}^K x'_i \mathbf{u}_i; \quad (11.14)$$

where  $\{\mathbf{u}_i\}_{i=1}^N$  are the principal components.

Therefore the lost information when we discard last  $N - k$  eigenvectors is given by:



$$\mathbf{x}' - \mathbf{x}'' = \sum_{i=K+1}^N x'_i \mathbf{u}_i;$$

It is possible to show [8] that the average square error  $\mathcal{E}$  on a dataset  $\Omega = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$  discarding last  $N - k$  eigenvectors is given by:

$$\mathcal{E} = \frac{1}{2} \sum_{i=1}^{\ell} \|\mathbf{x}'_i - \mathbf{x}''_i\|^2 = \frac{1}{2} \sum_{i=k+1}^{\ell} \lambda_i \quad (11.15)$$

where  $\{\lambda_i\}_{i=1}^N$  are the eigenvalues of the principal components.

#### 11.4.1 Nonlinear Principal Component Analysis

As we have seen PCA is not able to represent nonlinear components. In order to overcome this limitation, nonlinear algorithms have been proposed to get a nonlinear PCA. Among different possible approaches [48][51] to get a nonlinear PCA, the *autoassociative approach* is the most common one.

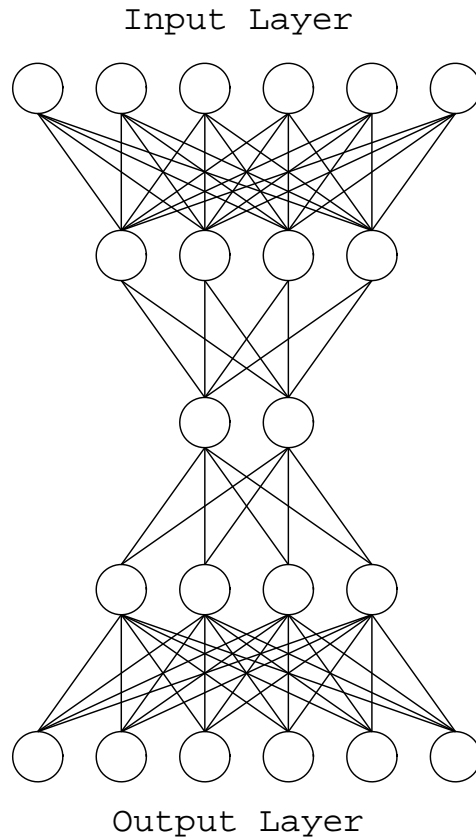
It can be shown [3] that an autoassociative three-layers neural network can only perform a linear PCA. Therefore if we want to perform a *nonlinear PCA* it is necessary to use neural network with a more complicated architecture, e.g. a five-layers neural network.

A neural net for the nonlinear PCA has a typical bottleneck structure, shown in Figure 11.3. The first (*input*) and the last (*output*) layer have the same number of neurons, while the remaining hidden layers have less neuron than the first and the last ones. The second, the third and the fourth layer are called respectively *mapping*, *bottleneck* and *demapping* layer. Mapping and demapping layers have usually the same number of neurons. Both mapping and demapping layer are formed by nonlinear units. The bottleneck layer (or *middle layer*) consists of linear units whose number  $m$  is lower than the original pattern dimensionality  $d$ . Each unit of the bottleneck layer represents a *nonlinear component* of data.

The targets used to train nonlinear PCA are simply the input vector themselves. Therefore each pattern is presented as both the input and as the target output. The network is trained with the backpropagation algorithm, minimizing the square error. As optimization algorithm, the *conjugate-gradient algorithm* [70] is generally used.

The number of the neurons of the bottleneck layer can be set up equal to the data dimensionality, if ID has been previously estimated by means of any ID estimation method (see Section 11.3). On the other hand, if we set up the the number of the neurons of the bottleneck selecting the one which minimizes the square error, the number itself can provide an ID estimate. Nonlinear PCA generally performs better than linear PCA as ID estimator [27].

Although nonlinear PCA is effective in several contexts, it presents some drawbacks. As underlined by Malthouse [61], the projections onto curves and



**Fig. 11.3.** A Neural Net for Nonlinear PCA

surfaces are suboptimal. Besides, NLPCA cannot model curves or surfaces that intersect themselves.

## 11.5 Independent Component Analysis

In Section 11.4 we saw that principal component analysis yields uncorrelated components. In this section we present a technique, the *independent component analysis (ICA)* [16][46], that yields *statistically independent components*. Our ICA description follows the same approach described in [43].

In order to introduce the ICA,<sup>3</sup> we consider the so-called *cocktail-party problem*. Suppose we have two people in a room who are speaking at the same time and two microphones located in different points of the room. We

<sup>3</sup> For sake of space, we do not describe the kernel version of ICA [2].

call  $s_1(t)$  and  $s_2(t)$  the signals detected at the time  $t$  by the two microphones. The two signals are linear combination of the sound signals  $u_1(t)$  and  $u_2(t)$  emitted by the two people, that is:

$$\begin{aligned}s_1(t) &= w_{11}u_1(t) + w_{12}u_2(t) \\ s_2(t) &= w_{21}u_1(t) + w_{22}u_2(t)\end{aligned}$$

where  $w_{ij}$  ( $i, j = 1, 2$ ) are unknown mixing parameters that depend upon the attenuations due to the distances of the microphones from two people.

Since mixing parameters are unknown the cocktail-party problem cannot be solved using classical numerical analysis methods. Nevertheless, the problem can be solved making some assumptions about the statistical properties of the sources  $u_1$  and  $u_2$ . We will show in the rest of the section that it is adequate to assume that the sources are statistically independent to solve the cocktail problem.

In order to define ICA, we generalize the cocktail problem introducing a *latent variable model*. We consider  $n$  linear mixtures  $s_1, \dots, s_n$  of  $n$  *independent components*  $u_1, \dots, u_n$  defined as follows:

$$\begin{aligned}s_1 &= a_{11}u_1 + \dots + a_{1n}u_n \\ &\dots \\ s_n &= a_{n1}u_1 + \dots + a_{nn}u_n\end{aligned}\tag{11.16}$$

We assume that in our model each mixture  $s_j$  is a random variable instead of a signal, as in the cocktail problem. For this reason, the time index in (11.16) does not appear.

We can rewrite the system (11.16) in a more concise and elegant way using a matrix notation. If  $\mathbf{s} = (s_1, s_2, \dots, s_n)$  and  $\mathbf{u} = (u_1, u_2, \dots, u_n)$  are the vectors whose components are, respectively,  $s_i$  and  $u_i$  and  $A$  the matrix whose elements are  $a_{ij}$  the equations (11.16) becomes:

$$\mathbf{s} = \mathbf{A}\mathbf{u}\tag{11.17}$$

The model described by Equation (11.17) is called the *independent component analysis* (or *ICA model*).

The ICA model is a *generative model*, that is a model that describes how the observed data (i.e  $\mathbf{s}$ ) can be generated by mixing *latent variables*  $u_i$ . The variables  $u_i$  are called *latent* since they cannot be directly observed. The mixing matrix  $A$  is assumed unknown. All we know is the observed vector  $\mathbf{s}$  and it has to be adequate to estimate  $A$  and  $u$ . ICA is strictly connected with the *blind source separation* (*BSS*) where we have some source signals, as in the cocktail party problem, and we do not know anything about the mixing matrix  $A$ , that the mixing process is *blind*. ICA is one of the most popular method for BSS.

The ICA model is based on the concept of *statistical independence*.

### 11.5.1 Statistical Independence

To introduce ICA model we need to define formally the concept of statistical independence.

**Definition 25 (Statistical Independence)** *Let  $u_1, u_2$  be two random variables. Let  $P(u_1, u_2)$  be their joint probability density function. Let  $P(u_1)$  and  $P(u_2)$  be the marginal density probability functions, respectively, of  $u_1$  and  $u_2$  defined as follows:*

$$P(u_1) = \int P(u_1, u_2) du_2; \quad P(u_2) = \int P(u_1, u_2) du_1.$$

*The variables are statistically independent if and only if the joint probability is given by:*

$$P(u_1, u_2) = P(u_1)P(u_2). \quad (11.18)$$

*Besides, if  $u_1, u_2$  are statistically independent then for any functions  $f(u_1)$  and  $g(u_2)$  the following equation holds:*

$$\mathcal{E}[f(u_1)g(u_2)] = \mathcal{E}[f(u_1)]\mathcal{E}[g(u_2)] \quad (11.19)$$

*where  $\mathcal{E}[\cdot]$  denotes the expectation operator.*

The definition of statistical independence can be immediately extended for any number of random variables larger than two.

Now, we recall the notion of *uncorrelatedness*.

**Definition 26** *Let  $u_1, u_2$  be two random variables. The variables  $u_1, u_2$  are uncorrelated if their covariance is null, that is:*

$$\mathcal{E}[u_1, u_2] = \mathcal{E}[u_1]\mathcal{E}[u_2] \quad (11.20)$$

*where  $\mathcal{E}[\cdot]$  denotes the expectation operator.*

Uncorrelatedness can be viewed as a weaker form of statistical independence [43]. Statistical independence implies uncorrelatedness (see Equation (11.19)). On the contrary, uncorrelatedness does not imply statistical independence (see Problem 11.9).

### 11.5.2 ICA Estimation

Having defined formally the concept of statistical independence, we can introduce the ICA model. Firstly, we can show that when the independent components are all Gaussian, we cannot solve ICA problem. For sake of simplicity, we consider the case of two sources  $u_1$  and  $u_2$  whose distribution is Gaussian. Besides, we assume that the mixing matrix  $A$  is orthogonal. The mixed variables  $s_1$  and  $s_2$  have joint probability density function given by:

$$p(s_1, s_2) = \frac{1}{2\pi} \exp\left(-\frac{s_1^2 + s_2^2}{2}\right) \quad (11.21)$$

It is possible to show [43] that the joint probability distribution function of any orthogonal transformation of  $s_1$  and  $s_2$  is given by (11.21) and that the mixed variables  $s_1$  and  $s_2$  are independent. Therefore when both sources are Gaussian we cannot estimate the mixing matrix  $A$  and solve the ICA problem. It is necessary to remark that the ICA problem can be solved when only one independent component is Gaussian.

In the ICA model we assume that the independent components are non-Gaussian. This assumption is in contrast with the statistical theory where random variables are usually assumed Gaussian. Besides, the *central limit theorem* claims that the distribution of a sum of independent random variables tends to the normal distribution under given conditions. The central limit theorem implies that each mixed observed variable  $s_i$  is closer to the normal distribution than each of the original independent components  $u_i$ . Being said that, we pass to estimate ICA model, that is to estimate the independent components.

We consider an observed data vector  $\mathbf{s}$  which is a mixture of independent components, that is it can be described by the matricial equation  $\mathbf{s} = A\mathbf{u}$ . For the sake of simplicity, we assume that the independent components  $u_j$  have the same probability distribution function. We define a linear combination of  $s_i$ , that is:

$$y = \mathbf{w}^T \mathbf{x} \quad (11.22)$$

where  $\mathbf{w}$  is a vector of parameters that has to be computed.

If  $\mathbf{w}$  is one of the row of  $A^{-1}$ , that is the inverse of the mixing matrix  $A$ , Equation (11.22) provides one of the required independent components  $u_i$ . Since the mixing matrix  $A$  is unknown, the vector cannot be exactly determined but only estimated.

Now we introduce a further variable  $z$  defined as  $\mathbf{z} = A^T \mathbf{w}$ . Equation (11.22) becomes:

$$y = \mathbf{w}^T \mathbf{s} = \mathbf{w}^T A \mathbf{u} = \mathbf{z}^T \mathbf{u}$$

Since, on the basis of the central limit theorem, the sum of independent variables  $u_i$  is more Gaussian than  $u_i$ ,  $z^T$  is more Gaussian than the independent components  $u_i$ . The variable  $z^T$  reaches the minimum of *Gaussianity* when  $z^T$  is equal to one of the independent components  $u_i$ . This observation motivates the following rule.

**Remark 2 (ICA Model Principle)** *To estimate the ICA model select the vector  $\mathbf{w}$  that maximizes the non-Gaussianity of  $\mathbf{w}^T \mathbf{s}$ .*

To use the ICA Model Principle we require a *measure of non-Gaussianity*. The simplest measure of non-Gaussianity is the *kurtosis*.

### Kurtosis

The *kurtosis* (or *fourth-order cumulant*) of a variable  $y$   $kurt(y)$  is defined by:

$$kurt(y) = \mathcal{E}[y^4] - 3(\mathcal{E}[y^2])^2.$$

where  $\mathcal{E}[\cdot]$  is the expectation operator.

Since for the Gaussian distribution  $\mathcal{E}[y^4]$  is equal to  $3(\mathcal{E}[y^2])^2$ , the kurtosis is zero for a Gaussian variable. On the other hand, most non-Gaussian variables have kurtosis nonzero. Random variables with positive kurtosis is called *sub-Gaussian* (or *platykurtic*). Whereas variables with positive kurtosis is called *super-Gaussian* (or *leptokurtic*). The non-Gaussianity is measured taken the absolute value of the kurtosis.

Although the kurtosis can be used for optimizing the non-Gaussianity, its usage presents many drawback. The main drawback of kurtosis is represented by its large sensitivity to outliers [38]. The value of the kurtosis may be notably affected by few data which can be noise or have poor representativity since whose values belong to the tail of the probability distribution function. Therefore kurtosis is not a robust measure of non-Gaussianity and other measures, for instance the *negentropy*, are advisable in most practical situations.

### Negentropy

The negentropy is a measure of non-Gaussianity strictly connected to the quantity, defined in in the information theory [19], called *entropy*. The entropy provides a measure of the casuality of the variable. The larger is the entropy of a variable the higher is its casuality.

The entropy for a discrete random variable  $Y$  is defined as follows:

$$H(Y) = - \sum_i P(Y = y_i) \log(Y = y_i)$$

where the  $y_i$  are the values that  $y$  can assume.

The definition of the entropy for discrete variables can be generalized for random variables  $\mathbf{y}$  with density  $g(\mathbf{y})$ . In this case, the entropy  $H(\mathbf{y})$  (usually called *differential entropy*) is given by:

$$H(y) = - \int g(\mathbf{y}) \log g(\mathbf{y}) d\mathbf{y}. \quad (11.23)$$

A fundamental result in information theory says that *among all random variables with equal variance, Gaussian variables have the largest entropy* [19]. This result implies that the entropy can be used to measure the non-Gaussianity of a variable.

Since it is useful a measure of non-Gaussianity that is nonnegative and zero for the Gaussian variables, it is preferable to use, as measure of non-Gaussianity, a modified version of the differential entropy, the so-called *negentropy* [43]. The negentropy  $J(\mathbf{y})$  of a variable  $\mathbf{y}$  is given by:

$$J(\mathbf{y}) = H(G) - H(\mathbf{y}) \quad (11.24)$$

where  $G$  is a Gaussian random variable having the same covariance matrix of  $\mathbf{y}$ .

For construction, the negentropy is always non-negative and is zero for the Gaussian variables. Moreover, negentropy is invariant for invertible linear transformations [16].

Negentropy is the optimal measure of non-Gaussianity under the point of view of the the information. Nevertheless, computing the negentropy is difficult since the estimation of the probability density function is required. Therefore it is convenient in the practical applications, replacing the negentropy with any of its approximations.

A popular method for approximating negentropy is based on higher-order moments. The approximation of negentropy  $\hat{J}(y)$  of a variable  $y$ , assumed to be of zero mean and unit variance, is given by:

$$\hat{J}(y) \sim \frac{1}{12} \mathcal{E}[y^3]^2 + \frac{1}{48} kurt(y)^2. \quad (11.25)$$

where  $kurt(\cdot)$  and  $\mathcal{E}[\cdot]$  are respectively the kurtosis and the expectation operator.

Since the right side of Equation (11.25) is function of the kurtosis, this approximation of negentropy inherits from the kurtosis its poor robustness towards the outliers.

An effective approximation of negentropy, based on maximum-entropy principle, has been proposed by Hyvärinen [40]. The Hyvärinen approximation of negentropy  $\hat{J}(y)$  of a variable  $y$ , assumed to be of zero mean and unit variance, is given by:

$$\hat{J}(y) \sim \sum_{i=1}^n \alpha_i \{ \mathcal{E}[K_i(y)] - \mathcal{E}[K_i(\mathcal{G})] \}^2 \quad (11.26)$$

where  $\alpha_i \in \mathbb{R}$ ,  $\mathcal{G}$  is a Gaussian variable of zero mean and unit variance and  $K_i(\cdot)$  are nonquadratic functions.

When  $n$  is equal to 1, the equation (11.26) becomes:

$$\hat{J}(y) \propto \{ \mathcal{E}[K(y)] - \mathcal{E}[K(\mathcal{G})] \}^2 \quad (11.27)$$

which is a generalization of the moment-based negentropy approximation. In fact, if we assume  $K(y) = y^4$  we get exactly Equation (11.25).

Choosing an appropriate form for  $G(\cdot)$  it is possible to obtain a more effective negentropy approximation than the one provided by (11.25). Suitable choices for the function  $G(\cdot)$  are given by:

$$G(x) = \frac{1}{\beta} \log \cosh \beta x \quad \beta \in [1, 2] \quad (11.28)$$

$$= -\exp\left(-\frac{x^2}{2}\right) \quad (11.29)$$

### 11.5.3 ICA by Mutual Information Minimization

A further approach for the ICA model estimation is based on the *mutual information minimization*. The mutual information  $\mathcal{I}(\mathbf{y})$   $n$  random variables  $(y_1, \dots, y_n)$  is defined as follows:

$$\mathcal{I}(\mathbf{y}) = \sum_{i=1}^n H(y_i) - H(\mathbf{y}). \quad (11.30)$$

where  $\mathbf{y} = (y_1, \dots, y_n)$ .

The quantity  $\mathcal{I}(\mathbf{y})$  is also called the *Kullback Leibler distance* between the probability density function  $g(\mathbf{y})$  and its independence version  $\prod_{j=1}^n g_j(y_j)$  where  $g_j$  is the probability density function of  $y_j$ .

The Kullback Leibler distance is always nonnegative and is null if and only if the variables are statistically independent. The mutual information has the property [19] that for any invertible linear transformation  $\mathbf{y} = W\mathbf{x}$ , we have:

$$\mathcal{I}(\mathbf{y}) = \sum_{i=1}^n [H(y_i) - H(\mathbf{x}) - \log |detW|] \quad (11.31)$$

where  $detW$  stands for the determinant of  $W$ .

If the variables  $y_i$  are uncorrelated and have unit variance then  $detW$  does not depend on  $W$  (see Problem 11.11), that is it can be viewed as a constant. Therefore Equation (11.31) becomes:

$$\mathcal{I}(\mathbf{y}) = C - \sum_{i=1}^n J(y_i) \quad (11.32)$$

where  $C$  is a constant and  $J(y_i)$  is the negentropy of the variable  $y_i$ .

Equation (11.32) underlines the connection between the mutual information and the negentropy.

The mutual information can be used for solving ICA model, since it is a measure of the independence of random variables. The approach, based on mutual information, is alternative to the approach based on the non-Gaussianity viewed in the section (11.5.2).

In the approach based on mutual information, we consider again the equation:

$$\mathbf{y} = W\mathbf{u}$$

and we look for a matrix  $W$  such that the mutual information of the observed data  $y_i$  is minimized. Equation (11.32) shows that the minimization of the mutual information is equivalent to maximizing the sum of the negentropies, that is the measures of non-Gaussianity, of the data  $y_i$  when  $y_i$  are constrained to be uncorrelated. Therefore the formulation of the ICA problem in terms of minimization of mutual information provides a further justification of the approach based on the minimization of non-Gaussianity.



We quote two other minor approaches, strictly connected, for the ICA model estimation, e.g. the maximum likelihood principle, the infomax principle. The *maximum likelihood method* [69] for the ICA problem is essentially equivalent to the approach based on the minimization of the mutual information [43]. The infomax approach is based on the *infomax principle* [6][65] which consists in maximizing the output entropy of a neural network having nonlinear output units. It has been proved [13][67] that the infomax principle is equivalent to the maximum likelihood principle and hence is similar to the method of the minimization of the mutual information.

Finally, we conclude our review on the methods of estimation of the ICA model underlining the connections on a methods, developed in statistics, called *exploratory projection pursuit* [29][28]. Exploratory projection pursuit (also called simply *projection pursuit*) is a technique for visualizing high-dimensional data. It has the aim of finding directions such that the data projections in those directions have interesting distributions, e.g. clusters. The Gaussian distribution is the least interesting one since fully determined by its mean and variance. On the other hand, non-Gaussian distributions are the most interesting since they presents structures such as clusters and long tails. In the exploratory projection pursuit some *projection indices* [29][28] have been proposed in order to measure the non-Gaussianity. Since a way for estimating the ICA model consists in measuring the non-Gaussianity, ICA can be viewed as a projection pursuit method. Moreover, the non-Gaussianity measures presented in Section 11.5.2 can be called, using a statistical terminology, projection indices.

#### 11.5.4 FastICA Algorithm

In this section we conclude our review on independent component analysis presenting one of the most popular algorithm for estimating ICA, the *FastICA algorithm*.

We have to point out that before applying any ICA algorithm on the data it is very useful, although not compulsory, to preprocess the data. The preprocessing usually consists in centering the data and the *whitening* (see Chapter IV). After centering the data, the data are zero-mean. After the whitening the data are white, that is the data components are uncorrelated and their variances are equal to one.

That being said, we pass to introduce the *FastICA algorithm* [42]. The FastICA algorithm is based on a fixed-point iteration scheme for finding a maximum of the non-Gaussianity of  $\mathbf{w}^T \mathbf{u}$ , where the measure of non-Gaussianity is given by (11.27), that is:

$$\hat{J}(y) \propto \{\mathcal{E}[K(y)] - \mathcal{E}[K(\mathcal{G})]\}^2$$

We denote with  $k(\cdot)$  and  $\mathcal{E}[\cdot]$ , respectively, the derivative of the function  $K(\cdot)$  and the expectation operator. The FastICA algorithm, whose derivation is omitted, for one independent component has the following steps:

1. Initialize the weight vector  $\mathbf{w}$
2. Compute

$$\hat{\mathbf{w}} = \mathcal{E}[\mathbf{u}k(\mathbf{w}^T \mathbf{u})] - \mathcal{E}[k'(\mathbf{w}^T \mathbf{u})]\mathbf{w}$$

where  $k'(\cdot)$  denote the derivative of the function  $k(\cdot)$

3. Update the vector  $\mathbf{w}$  replacing it with

$$\mathbf{w}_{new} = \frac{\hat{\mathbf{w}}}{\|\hat{\mathbf{w}}\|} \quad (11.33)$$

4. Compute  $i = \mathbf{w}_{new} \cdot \mathbf{w}$ .
5. If  $i = 1$  return the vector  $\mathbf{w}$  otherwise go to step 2.

The convergence of the algorithm is reached when the new and the old value of the vector  $\mathbf{w}$  has (roughly) the same direction. Therefore in practical applications, due to the presence of noise, the equality  $i = 1$  has to be replaced with  $i \sim 1$ .

The FastICA algorithm above described estimates only one of the independent components. If we want to estimate  $n$  (with  $n > 1$ ) independent components we have to use FastICA using  $n$  units with weight vectors  $\mathbf{w}_1, \dots, \mathbf{w}_n$ . If we want to obtain different independent components, avoiding that they converge at the same maxima, we have to decorrelate the outputs  $w_1^T, \dots, w_n^T$  after every iteration of the FastICA algorithm. Among several methods [41] [49] proposed for the decorrelation, we report the simplest method based on the *Gram-Schmid orthogonalization* [53]. In this method each independent component is estimated separately, that is the  $k^{th}$  independent component (with  $k \leq n$ ) is computed after the computation the first  $k - 1$  independent components. The  $k^{th}$  independent component is computed carrying out the FastICA algorithm for  $\mathbf{w}_k$  and after each algorithm iteration applying to the new vector  $\mathbf{w}_k$  the following transformation and renormalization :

$$\mathbf{w}'_k = \mathbf{w}_k - \sum_{i=1}^{k-1} (\mathbf{w}_k^T \cdot \mathbf{w}_i) \mathbf{w}_i \quad (11.34)$$

$$\mathbf{w}''_k = \frac{\mathbf{w}'_k}{\|\mathbf{w}'_k\|} \quad (11.35)$$

where  $w'_k$  and  $w''_k$  denote respectively the transformed and normalized vector.

Finally, we conclude quoting some properties [43] of the FastICA algorithms that make it preferable to other existing ICA algorithms. FastICA is faster than other ICA algorithm. It does not require parameters to be tuned unlike gradient-based ICA algorithms. FastICA estimates the independent components one by one and this property is quite useful when only some of independent components have to be estimated.

*FastICA* is a public domain software package, written in Matlab, developed by Hyvarinen that contains an implementation of the FastICA algorithms. It can be downloaded from: [http:// www.cis.hut.fi/projects/ica/fastica](http://www.cis.hut.fi/projects/ica/fastica).

## 11.6 Multidimensional Scaling Methods

*Multidimensional scaling (MDS)* [71][72] methods are dimensionality reduction techniques that tend to preserve, as much as possible, the distances among data. Therefore data that are close in the original data set should be projected in such a way that their projections in the new space, called *output space*, are still close. Among multidimensional scaling algorithms, the best-known example is *MDSCAL*, by Kruskal [54] and Shepard [76]. The criterion for the goodness of the projection used by MDSCAL is the *stress*. This depends only on the distances between data. When the *rank* order of the distances in the output space is the same as the rank order of the distances in the original data space, stress is zero. *Kruskal's stress*  $S_K$  is:

$$S_K = \left[ \frac{\sum_{i < j} [\text{rank}(d(\mathbf{x}_i, \mathbf{x}_j)) - \text{rank}(D(\mathbf{x}_i, \mathbf{x}_j))]^2}{\sum_{i < j} \text{rank}(d(\mathbf{x}_i, \mathbf{x}_j))^2} \right]^{\frac{1}{2}} \quad (11.36)$$

where  $d(\mathbf{x}_i, \mathbf{x}_j)$  is the distance between the data  $\mathbf{x}_i$  and  $\mathbf{x}_j$  and the  $D(\mathbf{x}_i, \mathbf{x}_j)$  is the distance of the projections of the same data in the output space. When the stress is zero a perfect projection exists. Stress is minimized by iteratively moving the data in the output space from their initially randomly chosen positions according to a gradient-descent algorithm. The intrinsic dimensionality is determined in the following way. The minimum stress for projections of different dimensionalities is computed. Then a plot of the minimum stress versus dimensionality of the output space is performed. ID is the dimensionality value for which there is a knee or a flattening of the curve. Kruskal and Shepard's algorithm presents a main drawback. The knee or the flattening of the curve could not exist. A MDS approach close to Kruskal and Shepard's one is the popular *Sammon's mapping* algorithm [74].

### 11.6.1 Sammon's Mapping

Sammon proposed to minimize a stress measure similar to Kruskal's one. The stress  $S_S$  proposed by Sammon has the following expression:

$$S_S = \left[ \sum_{i < j} \frac{(d(\mathbf{x}_i, \mathbf{x}_j) - D(\mathbf{x}_i, \mathbf{x}_j))^2}{d(\mathbf{x}_i, \mathbf{x}_j)} \right] \left[ \sum_{i < j} d(\mathbf{x}_i, \mathbf{x}_j) \right]^{-1} \quad (11.37)$$

Where  $d(\mathbf{x}_i, \mathbf{x}_j)$  is the distance between patterns  $\mathbf{x}_i$  and  $\mathbf{x}_j$  in the original data space and  $D(\mathbf{x}_i, \mathbf{x}_j)$  is the distance in the two- or three-dimensional output space. The stress is minimized by the gradient-descent algorithm.

Kruskal [55] demonstrated how a data projection very similar to Sammon's mapping can be generated from MDSCAL. An improvement to Kruskal's and

Sammon's methods has been proposed by Chang and Lee [15]. Unlike Sammon and Kruskal who move all points simultaneously in the output space to minimize the stress, Chang and Lee have suggested to minimize the stress by moving the points two at a time. In this way, it tries to preserve local structure while minimizing the stress. The method requires heavy computational resources even when the cardinality of the data set is moderate. Besides, the results are influenced by the order in which the points are coupled.

Several other approaches for MDS have been proposed. It is worth mentioning Shepard and Carroll's *index of continuity* [78], Kruskal's *indices of condensation* [56] and Kruskal and Carroll's parametric mapping [57]. Surveys of the classical multidimensional scaling methods can be found in [72][71][77].

Finally, it is worth mentioning the *curvilinear component analysis (CCA)* proposed by Demartines and Herault [20]. The principle of CCA is a *self-organizing* neural network performing two tasks: vector quantization of the data set, whose dimensionality is  $n$ , and a nonlinear projection of these quantizing vectors onto a space of dimensionality  $p$  ( $p < n$ ). The first task is performed by means of *SOM* [52]. The second task is performed by means of a technique very similar to MDS methods previously described. Since a MDS that preserve all distances is not possible, a cost function  $E$  measures the goodness of the projection. The cost function  $E$  is the following:

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} (d(\mathbf{x}_i, \mathbf{x}_j) - D(\mathbf{x}_i, \mathbf{x}_j))^2 F(D(\mathbf{x}_i, \mathbf{x}_j), \lambda) \quad (11.38)$$

where  $d(\mathbf{x}_i, \mathbf{x}_j)$  are Euclidean distances between the points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  of data space and  $D(\mathbf{x}_i, \mathbf{x}_j)$  are Euclidean distances between the projections of the points in the output space;  $\lambda$  is a set of parameters to set up and  $F(\cdot)$  is a function (e.g. a decreasing exponential or a sigmoid) to be chosen in an opportune way.

CCA seems to have very close performances to Shepard's MDS based on index of continuity [20].

## 11.7 Manifold Learning

*Manifold learning* [5][14] is a recent approach for nonlinear dimensionality reduction. Manifold learning algorithms are based on the idea that the intrinsic dimensionality of many data sets is small though each pattern can have several hundred of features. Therefore each pattern can be potentially described by only few parameters. Manifold Learning can be considered a generalization of the multidimensional scaling. In this section we first recall the mathematical concepts at the basis of the manifold learning and then we review the main manifold learning algorithms, that is *Isomap*, *locally linear embedding* and *Laplacian eigenmaps*.

### 11.7.1 The Manifold Learning Problem

Before introducing the manifold learning problem we have to recall some basic concepts of topology. Firstly, we define formally the concept of *manifold*. For this purpose we consider the semicircle shown in Figure 11.2. We have seen that the curve has intrinsic dimensionality equal to 1 though it is embedded in  $\mathbb{R}^2$ . Since the curve dimensionality is 1 the curve can be represented by means a unique variable. This concept can be formalized introducing the mathematical concept of *manifold*. We will say that the semicircle is a *one-dimensional manifold*. We pass to define formally the concept of manifold recalling the following definitions from the topology.

**Definition 27** A homeomorphism is a continuous function whose inverse is also a continuous function

**Definition 28** A  $d$ -dimensional manifold  $\mathcal{M}$  is a set that is locally homeomorphic with  $\mathbb{R}^d$ , i.e. for each  $m \in \mathcal{M}$  an open neighborhood around  $m$ , called  $N_m$  and a homeomorphism  $h : N_m \rightarrow \mathbb{R}^d$  exists.

The neighborhood  $N_m$  and the homeomorphism are respectively called the **coordinate patch** and **coordinate chart**. The image of the coordinate chart is called the **parameter space**.

The manifold is a very general concept. We are interested in the special case where the manifold is a subset of  $\mathbb{R}^d$ , that is  $\mathcal{M} \subset \mathbb{R}^d$ , and its dimensionality is such that  $d \ll N$ .

We introduce further definitions that can be useful in the rest of the section.

**Definition 29** A **smooth manifold** (or **differentiable manifold**) is a manifold such that each coordinate chart  $h$  is a **diffeomorphism**, i.e.  $h$  is differentiable and its inverse  $h^{-1}$  exists and is differentiable, too.

**Definition 30** An **embedding** of a manifold  $\mathcal{M}$  into  $\mathbb{R}^N$  is a **smooth homomorphism** from  $\mathcal{M}$  to a subset of  $\mathbb{R}^N$ .

That being said, we pass to define formally the *manifold learning problem*.

Let  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^N$  be a data set. Our goal is to reduce the number of features required to represent this data set. We assume that  $\mathcal{D}$  lies on a  $d$ -dimensional manifold  $\mathcal{M}$  embedded into  $\mathbb{R}^N$ , with  $d < N$ . Besides, we assume that the manifold is given by a single coordinate chart, that is equivalent to assume that the manifold is compact. The manifold learning problem is defined as follows.

**Problem 3 (Manifold Learning Problem)** Given a data set  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^N$  lying on a  $d$ -dimensional manifold  $\mathcal{M}$  described by a single coordinate chart  $h : \mathcal{M} \rightarrow \mathbb{R}^d$ , find  $\mathcal{D}' = \{\mathbf{y}_1, \dots, \mathbf{y}_\ell\} \in \mathbb{R}^d$  such that  $\mathbf{y}_i = h(\mathbf{x}_i)$  (for  $i = 1, \dots, \ell$ ).

The solution of this problem is called *manifold learning*.

Now we pass to review some manifold learning algorithms.

### 11.7.2 Isomap

*Isomap* [82], an acronym of *Isometric feature mapping*, is one of the most popular algorithm for manifold learning. Isomap algorithm can be viewed as an extension of MDS methods. Let  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathcal{M} \subset \mathbb{R}^N$  be a data set formed by data drawn by the manifold  $\mathcal{M}$ . Isomap has the aim of finding a coordinate chart that allows to project the data set in  $\mathbb{R}^d$ . Isomap assumes that an *isometric chart* exists, i.e. a chart that preserves the distances between the points. Therefore if two data points  $x_i, x_j \in \mathcal{M}$  have *geodetic distance*  $D_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j)$ , i.e. the distance along the manifold, then there is a chart  $h : \mathcal{M} \rightarrow \mathbb{R}^d$  such that:

$$\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\| = D_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j).$$

Besides, Isomap assumes that the manifold  $\mathcal{M}$  is smooth enough such that the geodetic distance between close points can be approximated by a line. Isomap uses the usual Euclidean distance between points to compute the geodetic distance between close points. On the contrary, Euclidean distance is not a good estimate of the geodetic distance between not close points, since the linear approximation becomes more and more inaccurate increasing the distance between points. In order to compute the geodetic distance Isomap builds a *neighborhood graph* in the following way. Isomap computes for each data point  $\mathbf{x}$  the set of its neighbors  $U(\mathbf{x})$  which can be composed in two different ways. In the first way the set of neighbors is formed by its  $K$  nearest neighbors, in the second way the set of neighbors is formed by all points whose distance is lower than  $\epsilon$ . The version of Isomap using the first way is called *K-Isomap*, whereas the version using the second way is the so-called  *$\epsilon$ -Isomap*. After the computation of the set of neighbors for each data point Isomap build a labelled graph  $\mathcal{G}$  over the data pattern of the data set  $\mathcal{D}$  where each pattern is represented by a vertex of  $\mathcal{G}$ . Besides, each vertex, corresponding to a given pattern  $\mathbf{x}$ , is connected to the vertices, corresponding to the patterns belonging the set of its neighbors  $U(\mathbf{x})$ , by a weighted edge. The weighted of the edge is given by the euclidean distances between the patterns representing the two vertices.

Then Isomap computes the geodetic distance  $D_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j)$  between all data points of  $\mathcal{D}$  by computing the shortest-path between the corresponding vertices on the graph  $\mathcal{G}$ . The shortest path can be computed by means of the *Dijkstra's* or the *Floyd's algorithm* [17]. At the end of this step, Isomap produces a matrix  $D_{\mathcal{M}}$  whose element  $D_{\mathcal{M}}(i, j)$  is given by the geodetic distance between the data points  $\mathbf{x}_i$  and  $\mathbf{x}_j$ , that is:

$$D_{\mathcal{M}}(i, j) = D_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j).$$

The final step of Isomap consists in applying a MDS algorithm (e.g. Sammon's mapping) constructing an embedding of the data in  $d$ -dimensional space which preserve as much as possible the geometry of the manifold. Unlike Laplacian

Eigenmaps and LLE, Isomap does not require that the dimensionality  $d$  of the manifold is a priori known.

Isomap can be summarized in the following steps:

- Take as input the data set  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^n$  and the parameter  $K$  (or alternatively the parameter  $\epsilon$ ).
- Compute the set of neighbors for each data point.
- Build the neighborhood graph
- Compute the shortest path graph given the neighborhood graph
- Make a  $d$ -dimensional embedding by means of a MDS algorithm.

The unique free parameter is  $K$  (or  $\epsilon$ ) that controls the size of the neighborhood. The parameter value is crucial [82] since the Isomap performances are strongly influenced by the size of neighborhood. Although in the original paper the parameter was tuned manually, a few techniques for tuning the parameter automatically are available [73].

Unlike most of manifold learning algorithms, Isomap guarantees theoretically the fidelity of the manifold reconstruction under given assumptions. If the manifold is compact, sampled everywhere, isometrically embedded in  $\mathbb{R}^d$  and the parameter space (i.e. the image of the chart) is convex then Isomap can reconstruct the manifold. This theoretical property justifies the increasing popularity of Isomap in the machine learning community.

A public domain software package implementing Isomap can be downloadable from: <http://isomap.stanford.edu>.

### 11.7.3 Locally Linear Embedding

*Locally linear embedding (LLE)* [75] is based on the idea of visualizing a manifold  $\mathcal{M}$  as a collection of overlapping coordinate patches. If the neighborhood sizes are small and the manifold is smooth the patches can be assumed roughly linear. Besides, the chart from the manifold  $\mathcal{M}$  to the lower dimensionality space  $\mathbb{R}^d$  is assumed to be approximatively linear on the patches. Therefore the idea underlying LLE consists in looking for local small patches, describing their geometry and finding a chart to  $\mathbb{R}^d$  that preserves the manifold geometry and is roughly linear. Besides, the local patches are assumed overlapped so that the local manifold reconstructions can be combined into a global one.

Let  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^N$  be a data set lying on a  $n$ -dimensional manifold  $\mathcal{M}$ . As in Isomap, the number of neighbors for a given pattern  $K$  is a parameter of the algorithm. Let  $U(\mathbf{x}_i)$  be the set of the  $K$ -nearest neighbors of the data point  $\mathbf{x}_i$ . The first step of LLE consists in modeling the manifold  $\mathcal{M}$  as a collection of linear patches and in estimating the geometry of the patches. The modeling is performed by representing each pattern  $\mathbf{x}_i$  as a convex combination of its nearest neighbors. The weights  $W_{ij}$  are obtained by the minimization of the following error:

$$\sum_{i=1}^{\ell} \|\mathbf{x}_i - \sum_{\mathbf{x}_j \in U(\mathbf{x}_i)} W_{ij} \mathbf{x}_j\|^2 \quad (11.39)$$

subject to

$$\sum_{\mathbf{x}_j \in U(\mathbf{x}_i)} W_{ij} = 1 \quad (11.40)$$

$$W_{ij} = 0 \quad \mathbf{x}_j \notin U(\mathbf{x}_i). \quad (11.41)$$

The matrix  $W_{ij}$  provides information about the local geometry of the patches describing the layout of the data points around  $\mathbf{x}_i$ . The constraint (11.41) makes explicit that LLE is a local method. On the other hand, the constraint (11.40) makes the weight matrix invariant to global translations, rotations and scalings (see Problem 11.12).

The constrained minimization problem can be solved using Lagrange multipliers. The vector of the reconstruction weights  $W_i$  for each pattern  $\mathbf{x}_i$  is given by:

$$\mathbf{W}_i = \frac{\sum_{k=1}^{\ell} C_{ik}^{-1}}{\sum_{l=1}^{\ell} \sum_{m=1}^{\ell} C_{lm}^{-1}} \quad (11.42)$$

where  $C$  is the local covariance matrix whose element  $C_{ij}$  is given by:

$$C_{jk} = (\mathbf{x} - \boldsymbol{\eta}_j)^T (\mathbf{x}_i - \boldsymbol{\eta}_k)$$

and  $\boldsymbol{\eta}_j$  and  $\boldsymbol{\eta}_k$  are neighbors of the pattern  $\mathbf{x}$ .

Since the vector  $\mathbf{W}_i$  corresponds to the  $i^{\text{th}}$  column of the matrix  $W$ , if we compute (11.41) for  $i = 1, \dots, \ell$  we obtain the whole reconstruction matrix  $W$ .

The vector  $\mathbf{W}_i$  defines the local geometry of the manifold around the pattern  $\mathbf{x}_i$ , i.e. the geometry of the neighborhood patch of  $\mathbf{x}_i$ .

The second step of LLE looks for a configuration in  $d$ -dimensions, i.e. the dimensionality of the parameter space, whose local geometry is described by the reconstruction matrix  $W$ . In LLE the dimensionality  $d$  must be a priori known or estimated before by means of an ID estimation algorithm.

The configuration can be obtained minimizing:

$$\sum_{i=1}^{\ell} \|\mathbf{y}_i - \sum_{j=1}^{\ell} W_{ij} \mathbf{y}_j\|^2 \quad (11.43)$$

with respect to  $\mathbf{y}_1, \dots, \mathbf{y}_\ell \in \mathbb{R}^d$ .

Equation (11.43) can be rewritten in the following matricial form:

$$Y^T M Y \quad (11.44)$$



where the element  $M_{ij}$  of the matrix is given by:

$$M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_{k=1}^{\ell} W_{ki} W_{kj}$$

and  $\delta_{ij}$  is the Kronecker symbol.

The matrix  $Y$  is subject to the following constraints:

$$\begin{aligned} Y^T Y &= \mathbb{I} \\ \sum_{i=1}^{\ell} Y_i &= 0 \end{aligned}$$

where  $\mathbb{I}$  is the identity matrix.

The equation (11.44) is a form of Rayleigh's quotient and is minimized by setting the column  $Y_i$  of the matrix  $Y$  equal to the  $d$  last nonconstant eigenvectors of  $M$ , i.e. the eigenvectors that correspond to  $d$  smallest nonzero eigenvalues of  $M$ .

LLE can be summarized as follows:

- Take as input the data set  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_{\ell}\} \in \mathbb{R}^n$ , the neighborhood size  $K$  and the manifold dimensionality  $d$ .
- Compute the reconstruction matrix  $W$ , whose column  $W_i$  is given by:

$$\mathbf{W}_i = \frac{\sum_{k=1}^{\ell} C_{ik}^{-1}}{\sum_{l=1}^{\ell} \sum_{m=1}^{\ell} C_{lm}^{-1}}$$

- Compute the low-dimensional embedding. Let  $M$  be a matrix defined by:

$$(\mathbf{I} - W)^T (\mathbf{I} - W)$$

Compute the matrix  $Y$  whose columns are given by the eigenvectors of the matrix  $M$  having eigenvalue nonzero.

- Return the  $(\ell \times d)$  matrix  $Y$ .

A public domain software package implementing LLE can be downloadable from: <http://basis.stanford.edu/HLLE>.

#### 11.7.4 Laplacian Eigenmaps

*Laplacian Eigenmaps* [5] is a manifold learning algorithm based on the *spectral graph theory*. Given a graph  $\mathcal{G}$  and a matrix of edge weights  $W$ , the *graph Laplacian*<sup>4</sup> is defined as

<sup>4</sup> This is not the unique definition of graph Laplacian, for other definitions see [64]

$$L = D - W$$

where  $D$  is a diagonal matrix whose elements  $D_{ii}$  are given by

$$D_{ii} = \sum_j W_{ij}.$$

The Laplacian provides informations about the graph, for instance the full connexion of a graph. In the Laplacian Eigenmaps the Laplacian provides local information about the manifold. In the Laplacian Eigenmaps a local similarity matrix  $W$  is defined. The matrix  $W$  measures how much points are close each other and can be defined in two different ways:

- $W_{ij} = 1$  if the pattern  $\mathbf{x}_j$  is one of the  $k$ -neighbors of  $\mathbf{x}_i$  (that is  $\mathbf{x}_j \in U(\mathbf{x}_i)$ ) and  $W_{ij} = 0$  otherwise.
- $W_{ij} = G(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$  if  $\mathbf{x}_j \in U(\mathbf{x}_i)$ , 0 otherwise.  $G(\cdot)$  is called the *Gaussian heat kernel*.

Laplacian eigenmaps uses the similarity matrix  $W$  to find the data points  $\mathbf{y}_1, \dots, \mathbf{y}_\ell$  which are the  $d$ -dimensional image of the points  $\mathbf{x}_1, \dots, \mathbf{x}_\ell$ . As in LLE, the manifold dimension  $d$  must be a priori known or estimated before by an ID algorithm.

Laplacian Eigenmaps minimize a cost function based on the following observation. If two data points have a large degree of similarity (i.e  $W_{ij}$  is large) then they are close each other on the manifold. Therefore their low-dimensional image should be close. This observation can be expressed to the following constrained minimization problem:

$$\min \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} W_{ij} (\mathbf{y}_i - \mathbf{y}_j)^2. \quad (11.45)$$

$$\begin{aligned} & \text{subject to} \\ & Y^T D Y = 1. \end{aligned} \quad (11.46)$$

where the  $j^{th}$  column of the matrix  $Y$  is given by  $\mathbf{y}_j$ .

The constraint (11.46) is necessary to avoid the trivial solution in which all  $\mathbf{y}_1, \dots, \mathbf{y}_\ell$  are given by the  $d$ -dimensional null vector (whose components are equal to zero).

The constrained minimization problem can be solved using Lagrange multipliers. It is possible to show that the constrained minimization problem is equivalent to the generalized eigenvalue problem, defined as follows:

$$LY = \lambda DY$$

whose solution is given by  $n$  last non-constant eigenvectors of  $M$ , i.e. the eigenvectors whose respective eigenvalue are nonzero.

Laplacian Eigenmaps can be summarized as follows:

- Take as input the data set  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \in \mathbb{R}^n$ , the neighborhood size  $K$  and the manifold dimensionality  $d$ .
- Set  $W_{ij} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$  if  $\mathbf{x}_j \in U(\mathbf{x}_i)$ , 0 otherwise.
- Let  $U$  be the matrix whose columns are given by the non-constant eigenvectors (whose respective eigenvalues are nonzero) of  $LY = \lambda DY$ .
- Return the  $(\ell \times n)$   $U$  matrix.

Although Laplacian eigenmaps is based on spectral graph theory is quite similar to LLE. This similarity has been theoretically shown by Belkin and Niyogi who proved that under certain assumptions LLE is equivalent to Laplacian Eigenmaps.

## 11.8 Conclusion

In this chapter we have presented some feature extraction methods and manifold learning algorithms. Firstly, we have discussed the curse of dimensionality in the framework of the function approximation theory. We have introduced the concept of data dimensionality describing some algorithms to estimate it. Then we have reviewed popular feature extraction methods such as principal and independent component analysis. We have also described statistical techniques, i.e. the multidimensional scaling algorithms, for data dimensionality reduction. Finally, we have introduced the problem of manifold learning describing main manifold learning algorithms.

## Problems

**11.1.** Consider the the *data set A* [39] of the *Santa Fe time series competition* were considered time series. Use the method of delays that is for each sample of the time series  $x(t)$  build a new vector  $X(t) = \{x(t), x(t-1), \dots, x(t-(d-1))\}$ , formed by the same pattern and its  $(d-1)$  antecedent samples. Let  $\Omega$  be the manifold generated by data points  $X(t)$ . Estimate the dimensionality of  $\Omega$  by means of the Grassberger-Procaccia algorithm. Assume in your experiments  $d = 10$ . Verify that your ID estimate fulfills *Eckmann-Ruelle inequality*. Compare your results with [12].

**11.2.** Consider a data set  $\Omega = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$  formed by vectors  $x_i \in \mathbb{R}^N$ . Project the data along the first  $k$ , with  $k < N$ , eigenvectors. Show that the average square error  $\mathcal{E}$  on the dataset  $\Omega =$  is given by:

$$\mathcal{E} = \frac{1}{2} \sum_{i=k+1}^{\ell} \lambda_i$$

where  $\{\lambda_i\}_{i=1}^N$  are the eigenvalues of the principal components (for the proof see [8]).

**11.3.** Consider the *singular value decomposition* defined as follows. Let  $A$  be a real  $m \times n$  matrix and  $l = \min(m, n)$ . There are orthogonal matrices  $U$  and  $V$  such that

$$A = UDV^T$$

where  $U \in \mathbb{R}^{m \times m}$ ,  $V \in \mathbb{R}^{n \times n}$ ,  $D = \text{diag}(\sigma_1, \dots, \sigma_l) \in \mathbb{R}^{m \times n}$ , that is if  $m > n$  has the form

$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots \\ 0 & \ddots & 0 \\ 0 & \cdots & \lambda_l \\ 0 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & 0 \end{bmatrix}$$

otherwise is the transpose. Prove that  $Y$  in (11.13) is given by  $Y = DV^T$ .

**11.4.** Implement PCA using a mathematical toolbox.

Test your implementation projecting *Iris Data* [26], that can be downloaded by <ftp.ics.uci.edu/pub/machine-learning-databases/iris>, along the major two principal components. Evaluate the information loss.

**11.5.** Repeat Problem 11.4 replacing Iris data with *Wisconsin Breast Cancer Database* [88] that can be downloaded by <ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin>. Compare the results.

**11.6.** Prove that the variances of the independent components cannot be determined.

**11.7.** Prove that the order of the independent components cannot be fixed. Namely we cannot label any of the independent components as the first one.

**11.8.** Prove that if  $u_1, u_2$  are two random variables statistically independent then for any functions  $f(u_1)$  and  $g(u_2)$  the equation (11.19) holds, that is:

$$\mathcal{E}[f(u_1)g(u_2)] = \mathcal{E}[f(u_1)]\mathcal{E}[g(u_2)]$$

**11.9.** Consider two random variables  $u_1, u_2 \in \mathbb{R}^2$  having the same probability of assuming any of the following values:  $\{(0, 0.5), (0, -0.5), (0.5, 0), (-0.5, 0)\}$ . Show that  $u_1$  and  $u_2$  are uncorrelated but not statistically independent.

**11.10.** Consider two independent random variables  $u_1, u_2$ . Prove that the kurtosis  $kurt(\cdot)$  satisfies the following properties:

$$\begin{aligned} kurt(u_1 + u_2) &= kurt(u_1) + kurt(u_2) \\ kurt(\alpha u_1) &= \alpha^4 kurt(u_1) \quad (\alpha \in \mathbb{R}) \end{aligned}$$

**11.11.** The mutual information  $\mathcal{I}(\mathbf{y})$ , for any invertible linear transformation  $\mathbf{y} = W\mathbf{x}$ , is given by:

$$\mathcal{I}(\mathbf{y}) = \sum_{i=1}^n [H(y_i) - H(\mathbf{x}) - \log |\det W|]$$

Prove that if the variables  $y_i$  are uncorrelated and have unit variance,  $\det W$  does not depend by  $W$ , that is it is a constant.

**11.12.** Prove that the reconstruction matrix  $W_{ij}$  of the LLE defined by:

$$\|\mathbf{x}_i - \sum_{\mathbf{x}_j \in U(\mathbf{x}_i)} W_{ij} \mathbf{x}_j\|^2$$

subject to:

$$\sum_{\mathbf{x}_j \in U(\mathbf{x}_i)} W_{ij} = 1; \quad W_{ij} = 0 \quad \mathbf{x}_j \notin U(\mathbf{x}_i)$$

is invariant to the global translations.



---

## References

1. Principal Component Analysis. *Principal Component Analysis*. Springer-Verlag, 1986.
2. F.R. Bach and M.I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3(1):1–48, 2002.
3. P. Baldi and K. Hornik. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.
4. A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.
5. M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.
6. A. Bell and T. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.
7. R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
8. C. Bishop. *Neural Networks for Pattern Recognition*. Cambridge University Press, 1995.
9. L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993.
10. J. Bruske and G. Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):572–575, May 1998.
11. F. Camastra. Data dimensionality estimation methods: A survey. *Pattern Recognition*, 36(12):2945–2954, December 2003.
12. F. Camastra and A. Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1404–1407, October 2002.
13. J.-F. Cardoso and B. Laheld. Equivalent adaptive source separation. *IEEE Transactions on Signal Processing*, 44(12):3017–3030, 1996.
14. G. Cayton. Algorithms for manifold learning. Technical report, Computer Science and Engineering department, University of California, San Diego, 2005.
15. C. L. Chang and R. C. T. Lee. A heuristic relaxation method for nonlinear mapping in cluster analysis. *IEEE Transactions on Computers*, C-23:178–184, February 1974.

16. P. Comon. Independent component analysis - a new concept? *Signal Processing*, 36(?):287–314, 1994.
17. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
18. J. Costa and A. O. Hero. Geodetic entropic graphs for dimension and entropy dimension in manifold learning. *IEEE Transactions on Signal Processing*, 52(8):2210–2221, 2004.
19. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.
20. P. Demartines and J. Herault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping in cluster analysis. *IEEE Transactions on Neural Networks*, 8(1):148–154, January 1997.
21. R. A. DeVore. Degree of nonlinear approximation. In *Approximation Theory, Vol. VI*, pages 175–201. Academic Press, 1991.
22. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley, 2001.
23. J. P. Eckmann and D. Ruelle. Ergodic theory of chaos and strange attractors. *Review of Modern Physics*, 57(3):617–659, 1985.
24. J. P. Eckmann and D. Ruelle. Fundamental limitations for estimating dimensions and lyapounov exponents in dynamical systems. *Physica*, D-56:185–187, 1992.
25. B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.
26. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
27. D. Fotheringham and R. J. Baddeley. Nonlinear principal component analysis of neuronal spike train data. *Biological Cybernetics*, 77(4):282–288, 1997.
28. J. H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–260, 1987.
29. J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–890, 1974.
30. K. Fukunaga. Intrinsic dimensionality extraction. In *Classification, Pattern Recognition and Reduction of Dimensionality, Vol. 2 of Handbook of Statistics*, pages 347–362. North Holland, 1982.
31. K. Fukunaga. *An Introduction to Statistical Pattern Recognition*. Academic Press, 1990.
32. K. Fukunaga and D. R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 20(2):165–171, 1976.
33. F. Girosi. Regularization theory, radial basis functions and networks. In *From Statistics to Neural Networks*, pages 166–187. Springer-Verlag, 1994.
34. F. Girosi and G. Anzellotti. Rates of convergence of approximation by translates. Technical report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.
35. P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica*, D9(1-2):189–208, 1983.
36. F. Hausdorff. Dimension und äusseres mass. *Math. Annalen*, 79(1-2):157–179, 1918.
37. A. Heyting and H. Freudenthal. *Collected Works of L.E.J Brouwer*. North-Holland Elsevier, 1975.



38. P. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.
39. U. Hübner, C. O. Weiss, N. B. Abraham, and D. Tang. Lorenz-like chaos in  $\text{nh}_3$ -fir lasers. In *Time Series Prediction. Forecasting the Future and Understanding the Past*, pages 73–104. Addison Wesley, 1994.
40. A. Hyvärinen. New approximations of differential entropy for independent component analysis and projection pursuit. In *Advances in Neural Information Processing Systems 10*, pages 273–279. MIT Press, 1998.
41. A. Hyvärinen. The fixed-point algorithm and maximum likelihood for independent component analysis. *Neural Processing Letters*, 10(1):1–5, 1999.
42. A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997.
43. A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.
44. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
45. L. K. Jones. A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regression and neural network training. *Journal of the Royal Statistical Society*, 20(1):608–613, March 1992.
46. C. Jutten and J. Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991.
47. D. Kaplan and L. Glass. *Understanding Nonlinear Dynamics*. Springer-Verlag, 1995.
48. J. Karhunen and J. Joutsensalo. Representations and separation of signals using nonlinear pca type learning. *Neural Networks*, 7(1):113–127, 1994.
49. J. Karhunen, E. Oja, L. Wang, R. Vigarío, and J. Joutsensalo. A class of neural networks for independent component analysis. *IEEE Transactions on Neural Networks*, 8(3):486–504, 1997.
50. B. Kégl. Intrinsic dimension estimation using packing numbers. In *Advances in Neural Information Processing 15*, pages 681–688. MIT Press, 2003.
51. M. Kirby. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*. John Wiley, 2001.
52. T. Kohonen. *Self-Organizing Map*. Springer-Verlag, 1995.
53. G. A. Korn and T. M. Korn. *Mathematical Handbook for Scientists and Engineers*. Dover, 1961.
54. J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.
55. J. B. Kruskal. Comments on a nonlinear mapping for data structure analysis. *IEEE Transaction on Computers*, C-20:1614, December 1971.
56. J. B. Kruskal. Linear transformation of multivariate data to reveal clustering. In *Multidimensional Scaling, vol. I*, pages 101–115. Academic Press, 1972.
57. J. B. Kruskal and J. D. Carroll. Geometrical models and badness-of-fit functions. In *Multivariate Analysis, vol. 2*, pages 639–671. Academic Press, 1969.
58. E. Levina and P. Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in Neural Information Processing 17*, pages 777–784. MIT Press, 2005.
59. Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transaction on Communications*, 28(1):84–95, 1980.
60. G. G. Lorentz. *Approximation of Functions*. Chelsea Publishing, 1986.

61. E. C. Malthouse. Limitations of nonlinear pca as performed with generic neural networks. *IEEE Transaction on Neural Networks*, 9(1):165–173, 1998.
62. B. Mandelbrot. *Fractals: Form, Chance and Dimension*. Freeman, 1977.
63. T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.
64. B. Mohar. Laplace eigenvalues of graphs: a survey. *Discrete Mathematics*, 109(1-3):171–183, 1992.
65. J.-P. Nadal and N. Parga. Nonlinear neurons in the low noise limit: a factorial code maximizes information transfer. *Networks*, 5(4):565–581, 1994.
66. E. Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 1993.
67. B. A. Pearlmutter and L. C. Parra. Maximum likelihood blind source separation: A context-sensitive generalization of ica. In *Advances in Neural Information Processing 9*, pages 613–619. MIT Press, 1997.
68. K. Pettis, T. Bailey, T. Jain, and R. Dubes. An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1(1):25–37, 1979.
69. D.-T. Pham, P. Garrat, and C. Jutten. Separation of a mixture of independent sources through a maximum likelihood approach. In *Proceeding EUSIPCO92*, pages 771–774, 1992.
70. W. H. Press, B. P. Flannery, S. A. Teukosky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1989.
71. A. K. Romney, R. N. Shepard, and S. B. Nerlove. *Multidimensional Scaling, vol. 2, Applications*. Seminar Press, 1972.
72. A. K. Romney, R. N. Shepard, and S. B. Nerlove. *Multidimensional Scaling, vol. 1, Theory*. Seminar Press, 1972.
73. O. Samko, A. D. Marshall, and P.L. Rosin. Selection of the optimal parameter value for the isomap algorithm. *Pattern Recognition Letters*, 27(9):968–979, 2006.
74. J. W. Jr. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transaction on Computers*, C-18(5):401–409, May 1969.
75. L. K. Saul and S. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, June 2003.
76. R. N. Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. *Psychometrika*, 27(3):219–246, June 1962.
77. R. N. Shepard. Representation of structure in similarity data problems and prospects. *Psychometrika*, 39(4):373–421, December 1974.
78. R. N. Shepard and J. D. Carroll. Parametric representation of nonlinear data structures. In *Multivariate Analysis*, pages 561–592. Academic Press, 1969.
79. L. A. Smith. Intrinsic limits on dimension calculations. *Physics Letters*, A133(6):283–288, 1988.
80. R. L. Smith. Optimal estimation of fractal dimension. In *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity vol. XII*, pages 115–135. Addison Wesley, 1992.
81. F. Takens. On the numerical determination of the dimension of an attractor. In *Dynamical Systems and Bifurcations, Proceedings Groningen 1984*, pages 99–106. Springer-Verlag, 1984.
82. J. B. Tanenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(12):2319–2323, December 2000.

83. J. Theiler. Lacunarity in a best estimator of fractal dimension. *Physics Letters*, A133(4-5):195–200, 1988.
84. J. Theiler. Statistical precision of dimension estimators. *Physical Review*, A41:3038–3051, 1990.
85. J. Theiler, S. Eubank, A. Longtin, B. Galdrikian, and J. D. Farmer. Testing for nonlinearity in time series: the method for surrogate data. *Physica*, D58(1-4):77–94, 1992.
86. G. V Trunk. Statistical estimation of the intrinsic dimensionality of a noisy signal collection. *IEEE Transaction on Computers*, 25(2):165–171, 1976.
87. P. J. Verveer and R. Duin. An evaluation of intrinsic dimensionality estimators. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 17(1):81–86, January 1995.
88. W. H. Wolberg and O. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences, U.S.A.*, 87(1):9193–9196, 1990.



Applications



## Speech and Handwriting Recognition

### What the reader should know to understand this chapter

- Hidden Markov models (Chapter 10).
- Language models (Chapter 10).
- Bayes decision theory (Chapter 3).

### What the reader should know after reading this chapter

- State-of-the-art in speech and handwriting recognition.
- Training of a language model.
- Software packages for the application of hidden Markov models.

### 12.1 Introduction

This chapter presents *speech* and *handwriting recognition*, i.e. two major applications involving the markovian models described in Chapter 10. The goal is not only to present some of the most widely investigated applications of the literature, but also to show how the same machine learning techniques can be applied to recognize data apparently different like handwritten word images and speech recordings. In fact, the only differences between handwriting and speech recognition systems concern the so-called *front-end*, i.e. the low-level processing steps dealing directly with the raw data (see Section 12.2 for more details). Once the raw data have been converted into sequences of vectors, the same recognition approach, based on hidden Markov models and  $N$ -grams, is applied to both problems and no more domain specific knowledge is needed. The possibility of dealing with different data using the same approach is one of the main advantages of machine learning, in fact it makes it possible to work on a wide spectrum of problems even in absence of deep problem specific knowledge.

Both speech and handwriting recognition have been investigated for several decades. The reason is that writing and speaking are two of the most common forms of human communication, then the possibility of converting spoken and handwritten data into digital texts can lead people to interact more naturally with computers. Moreover, huge amounts of information important for several activity domains are still stored under the form of handwritten documents (e.g. forms, letters, historical documents, etc.) and speech recordings (radio and television news, etc.).

This chapter focuses on systems recognizing unconstrained handwritten texts and conversational speech. Both problems require to apply not only HMMs to model the sequences of vectors extracted from the data, but also  $N$ -grams to provide a-priori probabilities for certain sequences of words being written or uttered. The above problems represent the most difficult challenge for current state-of-the-art systems. The recognition of less complex data like single handwritten or spoken words has been not only investigated, but it is also applied in real world problems like the recognition of handwritten addresses or spoken digits in cellular phones (see Section 12.8).

Chapter 10 has presented a tool for the creation of  $N$ -gram based language models (see Chapter 10) and this chapter will describe HTK,<sup>1</sup> the most commonly applied software package for training and testing hidden Markov models. The two packages are compatible with each other and can be used to build recognizer prototypes.

More extensive information about speech and handwriting recognition is available in both surveys [69][82][88] and monographies [5][39][84][72], for HMMs and  $N$ -grams, the reader can refer to Chapter 10 and references therein.

The rest of this chapter is organized as follows: Section 12.2 describes the structure of a recognition system. Section 12.3 presents the low-level processing aspects for both handwritten and spoken data. Section 12.4 focuses on HMM training issues. Section 12.5 shows the recognition process and the performance measures. Section 12.6 presents some results obtained on handwritten data. Section 12.7 shows results obtained by state-of-the-art speech recognition systems. Section 12.8 presents the major applications involving spoken and handwritten data.

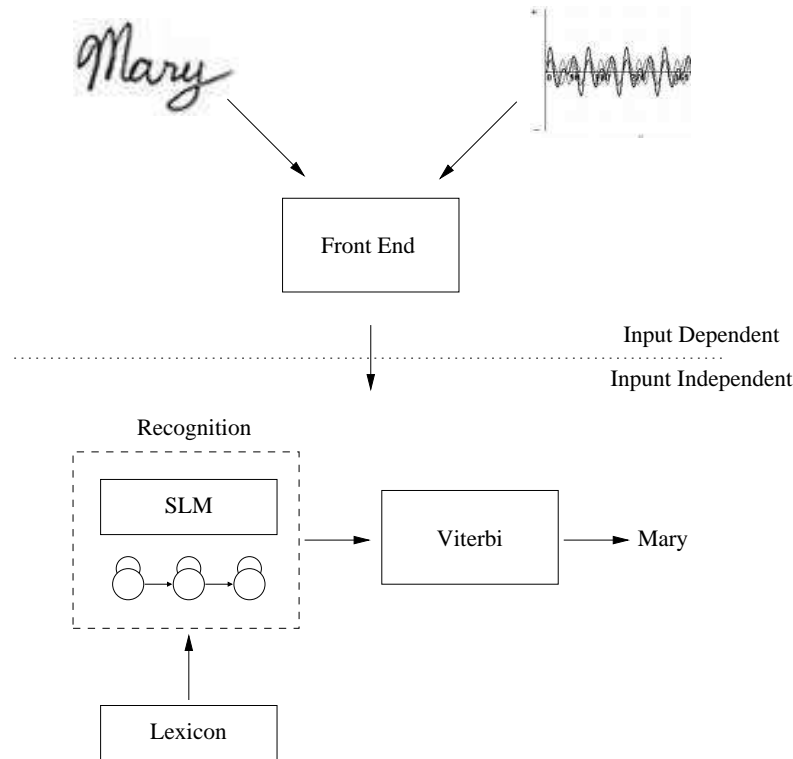
## 12.2 The General Approach

Figure 12.1 shows the general structure of speech and handwriting recognition systems. The horizontal dotted line splits the scheme into two major stages: the first is called *front end* and it converts the input data into a sequence of vectors (often called *observations* or *feature vectors*), the second is

---

<sup>1</sup> At the time this book is being written, the package can be downloaded at <http://htk.eng.cam.ac.uk>.





**Fig. 12.1.** Recognition system structure. The picture shows the general scheme of a recognition system. The vertical dotted line separates the feature extraction stage (which requires input dependent approaches) from the recognition process (which does not depend on the input).

called *recognition* and it converts the sequence of vectors into a digital word corresponding (hopefully) to the input image or utterance. The most important aspect of the split is that all steps that depend on the specific kind of data under examination (speech or handwriting) are concentrated in the front end. In other words, while the front end involves different algorithms when dealing with speech rather than handwriting, the recognition involves exactly the same machine learning techniques, namely hidden Markov models and  $N$ -grams (see Chapter 10), trained over different data.

The front end can be thought of as a function mapping the input data into a sequence of feature vectors. The main reason for the use of vector sequences rather than single vectors is that both handwritten words and spoken utterances are composed of smaller basic units, the letters in the first case and the phonemes (see Chapter 2) in the second case. Moreover, the use of single vectors for each word would lead to classification problems involving too many classes. In fact, realistic recognition problems require at least 20000 words in

the lexicon (see below) while few examples are available for each one of them (see the Zipf law in Chapter 10). The same problem does not apply to letters (at least for certain alphabets) and phonemes. In both cases around 40-50 classes are sufficient and a significant number of examples for each letter or phoneme can be collected using few thousands of words.

In both cases the sequence of vectors is obtained by first splitting the data into fragments and then by applying a feature extraction algorithm to each fragment. In the case of speech, the data is a soundwave (see Chapter 2), i.e. a signal  $s[n]$  containing physical measurements obtained at regular time steps. The fragments are obtained by isolating short signal windows (typically 30 ms long) shifted by 10-20 ms with respect to each other (see Section 2.5.3). In the case of handwriting, the data are images and there are two main ways to split the data: the first is to find points likely to be the frontier between two neighboring characters and then to split the data in their correspondence. The second is to have a fixed width window shifting from left to right by regular intervals and spanning the whole image.

The recognition is based on hidden Markov models and  $N$ -grams (see Chapter 10 for more details) and converts the sequence of vectors given as output by the front end into a digital text corresponding (if everything works correctly) to the actual written or spoken words. In general terms, the recognition identifies the sequence of words that best matches the sequence of vectors corresponding to the data. However, the set of the possible transcriptions is constrained by the *lexicon*, i.e. a predefined list of words accompanied by their transcriptions in terms of basic units (see Section 12.4). The output of the recognizer can contain only words belonging to the lexicon and this results into a major problem: if the raw data contain *out-of-vocabulary* (OOV) words, this automatically results into a transcription error. On the other hand, the lexicon provides an important constraint that limits, sometimes significantly, the number of hypotheses that the system has to take into account before providing the output. On average, the lower the number of words in the lexicon, the easier the recognition task. In fact, a limited number of hypotheses means that the computational burden is lower and that the risk of misclassifications due to ambiguities between similar words is reduced. For this reason, the lexicon size is one of the most important factors in defining the complexity of a recognition task (see Section 12.8).

The  $N$ -gram models (see Chapter 10 for more details) are necessary to deal with sentences and provide the a-priori probability of a certain sequence of words being written or uttered. The models are trained using large text corpora and the same model can be used for both speech and handwriting data as long as they use the same language.

In the next sections, all processing steps will be explained in more detail.

## 12.3 The Front End

This section describes the front end of both speech and handwriting recognition systems. The main difference between speech and handwriting is that in the first case there are few theoretically grounded methods that are applied by most systems, while in the second case, the front end is rather based on empirical algorithms and each system presented in the literature uses different techniques. However, despite the wide spectrum of different approaches there is at least consensus on which tasks must be carried out. For this reason, the description of the handwriting front end will focus on the description of such operations, while the description of the speech front end will focus on the *Mel frequency cepstrum coefficients* extraction, maybe the most common front end in speech recognition systems.

### 12.3.1 The Handwriting Front End

The first step in the handwriting front end is so-called *preprocessing* which takes as input the raw data images and gives as output a binary image where only the words to be recognized are displayed. This step depends on the data, e.g. in bank checks it is often necessary to remove security background textures; in forms it can be necessary to remove rulers and cases used to guide the writers; in the case of historical documents it can be necessary to remove humidity traces or other effects. All above tasks can be performed using common image processing techniques such as binarization, texture recognition, and filtering, but the high variability in the input data can require more specific and refined techniques.

The second step is so-called *normalization*, i.e. the removal of the variability unnecessary to the recognition process. In the case of handwritten words, the normalization addresses *slope* and *slant*. The first is the angle between the horizontal direction and the direction of the line on which the word is aligned, the second is the angle between the vertical direction and the direction of the strokes supposed to be vertical in an ideal model of handwriting (see Figure 12.2). The literature proposes different techniques to normalize handwritten words, but none of them appears to overperform the others. On the other hand, there is consensus about the effectiveness of the normalization in improving the recognition systems performance.

After the normalization, the handwritten images can be converted into vector sequences. This step is called *feature extraction* and it involves two tasks, the first is the segmentation into word subunits, the second is the conversion of each subunit into a single vector. The segmentation is performed using two major approaches. The first tries to split the data in correspondence of points expected to delimit basic patterns that can be grouped into few classes (the so-called *atoms*). The most common techniques split the data in correspondence of minima and maxima of the word contour, on the sides of loops, etc. The second approach is based on the so-called *sliding window*,

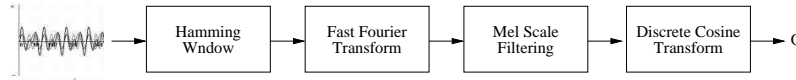


**Fig. 12.2.** Handwritten word normalization. The upper picture shows slant and slope, the central picture shows the so-called *core region* (the region containing the character bodies), and the lower picture shows the result of the normalization.

i.e. a fixed width window shifting by a predefined number of pixels and spanning the image from left to right (in general the patterns identified by two consecutive window positions overlap each other).

After the segmentation, a feature extraction process is applied to each subunit. The literature proposes a wide range of techniques (see [85] for an extensive survey), the most common features try to detect structural characteristics (i.e. loops, holes, vertical strokes, etc.), account for the distribution of foreground pixels in the bounding box containing the subunit, provide a measure of the alignment of foreground pixels along a set of predefined directions, etc.

In general, the front end of the handwriting system is based on empirical considerations that do not involve any principled or theoretically justified approach. On the other hand, the major efforts are typically made at the recognition level, where a correct application of machine learning approaches



**Fig. 12.3.** MFCC extraction block diagram.

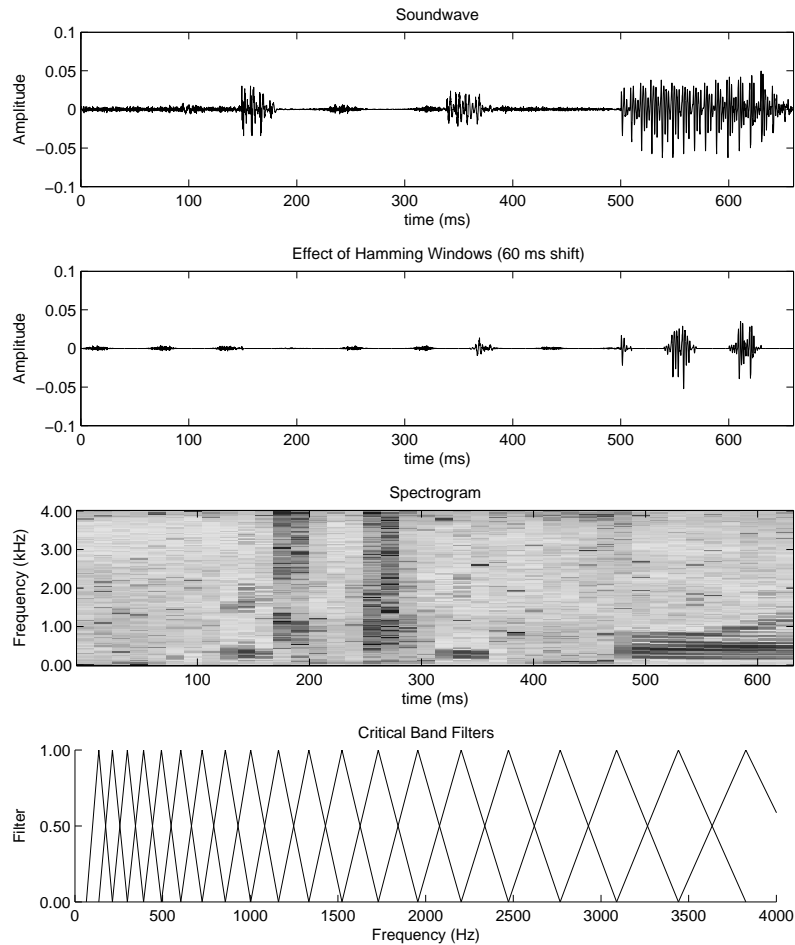
can make a significant difference in terms of recognition rate (percentage of words correctly recognized).

### 12.3.2 The Speech Front End

As opposed to the front end of handwriting, the speech front end is based on signal processing methods and few techniques are used by most of the recognition systems. This section focuses on the *Mel frequency cepstrum coefficients* (MFCC) extraction, which is based on techniques shown in other parts of this book (see in particular Chapter 2 and Appendix B). It is one of the most widely applied speech processing techniques. Other popular methods, e.g. the *linear prediction coefficients* (LPC), can be found in specialized monographies [39].

Figure 12.3 shows the block diagram of the MFCC extraction, the first step is the application of a Hamming window (see Section 2.5.3) to the signal. This step corresponds to the segmentation of the handwriting images and the window width is typically fixed at 30 ms. The shift between two consecutive window positions is in general 10 ms. Such values represent a good tradeoff between the need of being short enough to detect phonemes boundaries and the need of being long enough to avoid local fluctuations. Both parameters have been validated through decades of experiments and, although not supported by any theoretic argument, have been shown empirically to give good results. The effect of the Hamming window can be observed in Figure 12.4, the first two plots from above show the raw signal and its convolution with 30 ms wide windows shifted by 50 ms. Each window isolates a segment of the raw signal which is then used for the following steps of the processing.

The second step of the MFCC extraction is the application of the Fourier Transform (see Appendix B) to each segment. The number of retained coefficients is 129, another parameter that has no theoretic support, but it has been shown to be effective through extensive empirical validation. The result is that the *spectrum* of the signal, i.e. the distribution of the energy at different frequencies is available at each window position. The graphical representation of such an information is called *spectrogram* and it is depicted in the third plot of Figure 12.4, the horizontal axis corresponds to the time, while the vertical one corresponds to the frequencies. A simple observation of the spectrogram shows that the characteristics of the signal are constant for certain time intervals and change suddenly to reach a different configuration that remain stable in the following time interval. The stability intervals roughly correspond to the different phonemes, i.e. to the different articulator configurations used in the voicing process (see Chapter 2 for more details).



**Fig. 12.4.** MFCC extraction. From the top to the bottom the plot show the original signal, some of the time segments extracted with the Hamming windows, the spectrogram and the Mel filters.

The spectrogram provides rich information about the signal, but it cannot be easily handled. It is necessary to select only part of its content. This is the goal of the following step in the MFCC extraction process. There is physiological evidence that humans cannot distinguish between frequencies belonging to the same *critical band* (see Chapter 2), i.e. frequency intervals centered around the so-called *critical frequencies*. For this reason, it is possible to sum the energies of the frequencies falling into the same critical band. As a result, each column of the spectrogram can be *summarized* by around 20 values (another empirical parameter) accounting for the total energy in a given critical band. This requires first to identify the critical frequencies and

this is done by using the *Mel scale* (hence the name Mel frequency cepstrum coefficient) introduced in Chapter 2. The critical frequencies are uniformly distributed along the Mel scale and the critical bands are centered around them. The width of the bands is set empirically with the only constraint that neighboring bands must have some degree of overlapping. The lowest plot in Figure 12.4 shows the filters centered around the critical frequencies (which are equispaced along the Mel scale, but not in the natural frequencies shown in the plot) and getting more and more coarse at higher frequencies (the amplitude of the filters is the same along the Mel scale, but the plot shows the filters in the natural frequencies domain).

At this point of the extraction process, the original signal is converted into a sequence of 20-22 dimensional vectors where each component accounts for the energy in a critical band. The last operation is the application of a *discrete cosine transform* (see Appendix B) to such vectors with the goal of *decorrelating* the data, i.e. of transforming the data so that the covariance between different components is null. Only the first coefficients of the DCT (in general 12) are retained and the resulting vector is the result of the front end process. The use of the DCT is at the origin of the name *Cepstrum*. In fact, the DCT can be interpreted as an inverse FT and, since it is applied to the *spectrum*, it leads to the *ceps-trum*. Although born as a joke, the name is used still today.

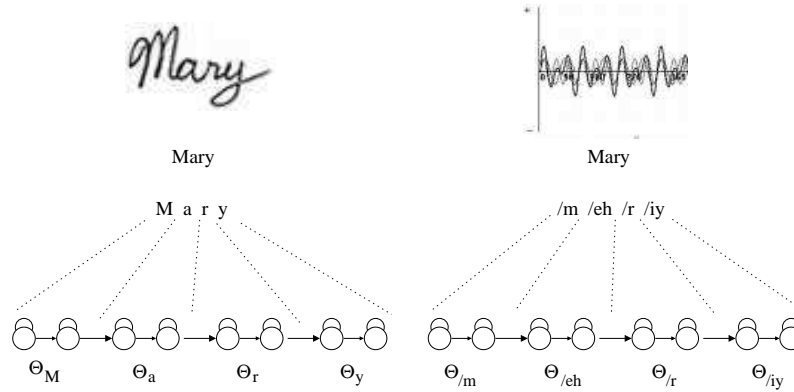
## 12.4 HMM Training

This section focuses on the training of hidden Markov models and  $N$ -grams for the recognition of speech and handwriting. While in the previous sections we had to distinguish between the two kinds of data, from now on both handwriting and speech can be addressed exactly in the same way. The training techniques and the use of the trained models in the recognition process are independent of the data and no more distinction will be made unless necessary. The next sections focus on the preparation of training material and lexicon and on the practical details of HMM training. For each operation involving the HMMs, we provide the HTK commands necessary to perform the task.

### 12.4.1 Lexicon and Training Set

The Baum-Welch algorithm, i.e. the expectation-maximization technique used to train the HMMs, has been described in Chapter 10. Here we focus on the practical details (including the use of HTK) of the HMM training in speech and handwriting recognition.

The first element necessary for the training is the training set, i.e. a set of examples (handwriting images or spoken utterances) like those shown in Figure 12.5, for which the transcription is available. The second element needed for the training is the lexicon, i.e. the list of unique words that the recognizer



**Fig. 12.5.** The lexicon. The picture shows (from top to bottom) the raw data, the transcription (*Mary* for both speech and handwriting), the codification (*M a r y* for handwriting and */m /eh /r /iy* for speech), and the word model obtained by concatenating single letter or phoneme models ( $\Theta_x$  is the HMM corresponding to letter or phoneme  $x$ ).

can give as output with their corresponding codification in terms of subunits. Figure 12.5 shows the difference between the data, the transcription, the codification in terms of subunits and the word models. The figure shows that the transcription is the same for both speech and handwriting, but the codification changes depending on the data: in the case of the handwriting, the subunits are letters, then the word *Mary* is coded with the sequence of the letters *M a r y*, in the case of the speech, the word is coded with the sequence of phonemes */m /eh /r /iy* corresponding to the sounds produced when uttering *Mary*. The lexicon contains, for each word, both the transcription and the codification. While the codification in terms of letters is unique, the codification in terms of phonemes can change. In fact, there are several sets of phonemes for a given language (the phonemes are manually defined and there is no general agreement on the best phoneme set). Moreover, given a phoneme set, there can be more than one way of pronouncing a word, then more than one valid codifications. Despite the above details, the important point is that the lexicon provides for each word a sequence of symbols belonging to a predefined set that cannot be changed without changing the recognition system.

The codification enables us to clarify a first important point when we say that we use HMMs for the recognition: there is one model for each subunit, i.e. for each letter or for each phoneme, then word models are built as a concatenation of subunit models (see bottom of Figure 12.5). This is important because it enables us to model any word using a small set of HMMs (letters in latin alphabet are 26 and the phoneme sets contain around fourty elements). In the figure,  $\Theta_x$  denotes the HMM corresponding to symbol  $x$  and the word model  $\Theta_w$  is obtained by concatenating the subunit models composing word



$w$  (there is ambiguity between  $w$  as a word symbol and  $w$  as a  $w$  letter symbol, but this should not create problems in the following).

### 12.4.2 Hidden Markov Models Training

The first problem to be addressed is the selection of the model topology. In speech and handwriting recognition, the topology is typically left-right (see Chapter 10) to account for spatial and temporal evolution of written and spoken words respectively. The second choice to be made is the emission probability function. The most commonly applied function is the mixture of Gaussians (see Chapter 10) and this is the case also in this chapter. This is the reason why the last step of the MFCC extraction is a DCT aiming at the decorrelation of the data. In fact, when the covariance of different components is null, the covariance matrices of the Gaussians can be diagonal and this spares a large number of parameters. In this way, given a certain amount of material, the training is more effective.

At this point the training can start and the models must be initialized. The most common approach is the *flat initialization*: all transition probabilities that must be different from zero are set to a uniform distribution and means and variances of all Gaussians appearing in the models are set to the global mean and variance of the training data at disposition. This task is performed by HTK using the following command:

```
HCompV hmmFile listTrainingFiles
```

where `hmmFile` is the file containing the models (see the HTK manual [94] for the format) and `listTrainingFiles` is the list of the files containing the feature vectors extracted from the training examples (the function has several options enabling one to control finer details).

After the initialization, the actual training can start. For each training example, a different HMM is built by concatenating the subunit models corresponding to the words the example contains. At this point, the Baum-Welch algorithm can be implemented through the following three steps that are reiterated until a stopping criterion (see below) is met:

1. The model corresponding to each training sample is aligned with the sequence of vectors using the Viterbi algorithm. This leads to a segmentation that associates each data segment to a specific subunit model.
2. Each training sample is processed in order to count the number of times a feature vector corresponds to a given state in a given model, the number of times a transition between two given states in each model takes place, compute the means and the variances of the feature vectors corresponding to each state in each model.
3. Counts, means and variances estimated in the second step are used to obtain new estimates of initial state probabilities, transition probabilities, means and variances of Gaussians in the mixtures, coefficients of the Gaussians in the mixtures.

The process can be stopped when a certain number of iterations has been reached or when the variation between two following estimates falls below a predefined threshold.

The process outlined above is called *embedded reestimation* because the subunit models are trained as a part of a larger model rather than as separate entities. The advantage is not only that this is a more realistic situation (letters and phonemes are always part of a word), but it enables one to avoid the manual segmentation of the samples into subunits through an expensive and time consuming manual work.

The embedded reestimation can be performed using the following HTK command:

```
HERest hmmFile listTrainingFiles,
```

see above for the meaning of the function arguments (the function options enable to control finer details of the training).

At the end of the training, the parameters of each subunit model are set to the values maximizing the likelihood of the training data. The training must be performed only once for a given system, provided that the training data are representative of the data to be encountered in the application setting. The next sections show how the models obtained during the training are used to perform the recognition.

## 12.5 Recognition and Performance Measures

This section focuses on the actual recognition process, i.e. on the use of the models described above to transcribe data, and on the way the performance of a recognition system is measured.

### 12.5.1 Recognition

The recognition can be formulated as the problem of finding the word sequence  $\hat{W} = \{\hat{w}_1, \dots, \hat{w}_N\}$  (where  $w_i$  belongs to the dictionary  $V$ ) maximizing the a posteriori probability  $p(\Theta_W|O)$ , where  $O = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$  is the vector sequence extracted from the data. Following the Bayes theorem (see Chapter 5):

$$\hat{W} = \arg \max_W \frac{p(O|\Theta_W)p(W)}{p(O)} \quad (12.1)$$

and, since  $O$  is constant during the recognition, the above equation can be rewritten as:

$$\hat{W} = \arg \max_W p(O|\Theta_W)p(W). \quad (12.2)$$

The last equation shows that the a-posteriori probability of a word sequence is estimated through the product of two terms: the first is the likelihood

$p(O|\Theta_W)$  of the vector sequence  $O$  given the model corresponding to the word sequence  $W$ , the second is the a priori probability  $p(W)$  of the word sequence  $W$  being written/uttered. The problem is then how to estimate the two terms for a given word sequence  $W$ .

The likelihood is estimated using the HMMs obtained during the training. Given a word sequence  $W$  and a vector sequence  $O$ ,  $p(O|\Theta_W)$  is in fact estimated using the Viterbi algorithm (see Chapter 10) which provides both the likelihood estimate (the highest possible value given  $O$  and  $W$ ) and the *alignment* of the model with the data, i.e. it attributes each vector in the sequence to one of the HMMs composing the subunit sequence corresponding to  $W$ . This is interesting because it gives the segmentation of the data into words and letters or phonemes as a side product of the recognition process. The probability  $p(W)$  is estimated using the  $N$ -gram models trained over a text corpus independent of the data to be recognized (see Chapter 10). Once the product  $P(O|\Theta_W)p(W)$  has been estimated for all possible sequences, the recognition simply gives as output the word sequence  $W$  corresponding to the highest value.

This approach leaves open an important problem: as the size of the lexicon increases, the number of possible word sequences becomes quickly high and the computational burden needed to align each sequence model with a given observation sequence becomes too heavy. The approaches dealing with such a problem are out of the scope of this book, but the interested reader can refer to [4][30] and references therein.

### 12.5.2 Performance Measurement

The performance of a recognition system is measured through the percentage of correctly transcribed words, called *word recognition rate* (WRR), or through its complement, i.e. the percentage of words uncorrectly transcribed, called *word error rate* (WER). The two measures are related through the following equation:

$$WER = 100 - WRR \quad (12.3)$$

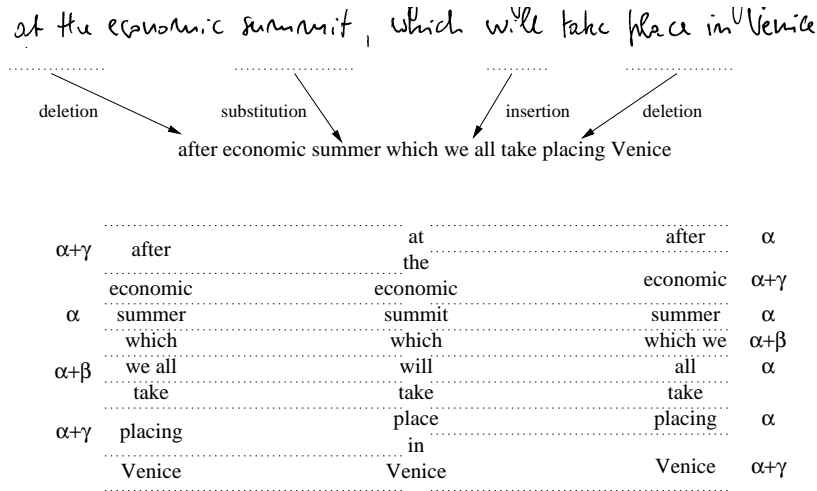
and are thus equivalent (in this chapter we will use the WRR).

For the recognition of single words, the computation of the WRR is straightforward; in fact, it simply corresponds to the percentage of test samples that have been correctly transcribed,<sup>2</sup> but the problem is more difficult for the recognition of word sequences. In fact, in this case a recognizer can perform three kinds of errors (see Figure 12.6) [40][42]:

*Substitution* The position of a word in the output sentence is correctly identified, but the transcription does not correspond to the word actually written/uttered (word *summit* in Figure 12.6).

---

<sup>2</sup> In this case, the decoding takes into account the fact that each sample corresponds to a single word and does not try to align the data with more than one word. This avoids deletion and insertion errors that are explained in the following.



**Fig. 12.6.** Handwritten word normalization. The upper picture shows slant and slope, the central picture shows the so-called *core region* (the region containing the character bodies), and the lower picture shows the result of the normalization.

*Insertion* A single word is split into two or more words and the result is not only a substitution, but also the insertion of one or more nonexisting words (word *will* in Figure 12.6).

*Deletion* Two or more words are transcribed as a single word and the result is not only a substitution, but also the loss of an existing word (words *at the* and *place in* in Figure 12.6).

Moreover, there are different ways of aligning the automatic transcription of a sentence with the groundtruth (i.e. the actual data transcription) and this requires to have a criterion to select the alignment to be used for the performance measurement (see Figure 12.6). The reason is that groundtruth and automatic transcription do not necessarily contain the same number of words. More than one word of the groundtruth can be associated to the same word of the transcription (leading to a deletion) or vice versa (leading to an insertion).

The most common approach is to associate a penalty with each alignment and to select the alignment with the lowest penalty (Figure 12.6 shows two alignments and respective penalties). The penalty for a given alignment is obtained as follows:

$$P = \alpha S + \beta I + \gamma D \quad (12.4)$$

where  $\alpha$ ,  $\beta$  and  $\gamma$  are coefficients to be set empirically,  $S$  is the number of substitutions,  $I$  is the number of insertions and  $D$  is the number of deletions. The *National Institute of Standards and Technologies* (NIST), which promotes the comparison between results obtained in different groups by providing common benchmarks, proposes to set  $\alpha = 4$  and  $\beta = \gamma = 3$ , while the HTK package

proposes  $\alpha = 10$  and  $\beta = \gamma = 7$ . The coefficient values do not affect significantly the resulting WRRs, but it is important to use the same coefficients when comparing different results.

Once the alignment has been selected, the WRR is estimated as follows:

$$WRR = \frac{N - D - S}{N} \times 100\% \quad (12.5)$$

where  $N$  is the total number of words in the groundtruth transcription. The above expression ignores the insertion errors (this does not affect the number of correctly recognized words), then a further measure, called *Accuracy* has been defined that try to better account for the matching between groundtruth and automatic transcription:

$$A = \frac{N - D - S - I}{N} \times 100\%. \quad (12.6)$$

There are no common criteria to decide whether to use  $A$  or  $WRR$ . In general, the  $A$  is a better measure when the goal is to obtain a transcription as close as possible to the groundtruth, while the WRR is useful when the goal is to transcribe correctly as many words as possible, e.g. in call routing.<sup>3</sup> However,  $A$  and WRR are strongly correlated and systems with high WRR have high  $A$  and viceversa.

The HTK tool provides a routine that computes the different performance measures given the groundtruth data and the automatic transcriptions:

```
HResults hmmList recFile
```

where `hmmList` is the list of the HMMs used by the recognizer (letter models for handwriting and phoneme models for speech) and `recFile` is the file containing the recognizer output. Several options enable to set the above mentioned parameters and to give the path to the files containing the groundtruth transcriptions of the data.

## 12.6 Recognition Experiments

This section focuses on practical issues involved in the recognition of unconstrained texts. The experiments involve handwritten data as an example, but all the considerations apply also to speech recognition experiments. The first dataset we use will be referred to as the *Cambridge* database and it is composed of a text written by a single person.<sup>4</sup> The Cambridge database was

<sup>3</sup> Call routing is the problem of automatically finding an operator capable of addressing the needs expressed by a person contacting a call center. In this case, a perfect transcription is not necessary; the only important thing is to recognize the few keywords identifying the user needs and the right operator

<sup>4</sup> The data is publicly available and it can be downloaded at the following ftp address: <ftp.eng.cam.ac.uk/pub/data>.

originally presented in [76] and contains 353 handwritten text lines split into training (153 lines), validation (83 lines) and test (117 lines) sets. The lines are kept in the same order as they were written to reproduce a realistic situation where the data available at a certain time are used to obtain a system capable of recognizing the data that will be written in the future. The second dataset we use is composed of pages written by several persons, it can be obtained at University of Bern [61][95] and it will be referred to as the *IAM* database. It is split into training, validation and test sets containing 416, 206 and 306 lines respectively. The data is split in such a way that the writers represented in the training set are not represented in the test set. This is assumed to reproduce a realistic situation where the data produced by a certain number of writers is used to recognize the data written by other persons.

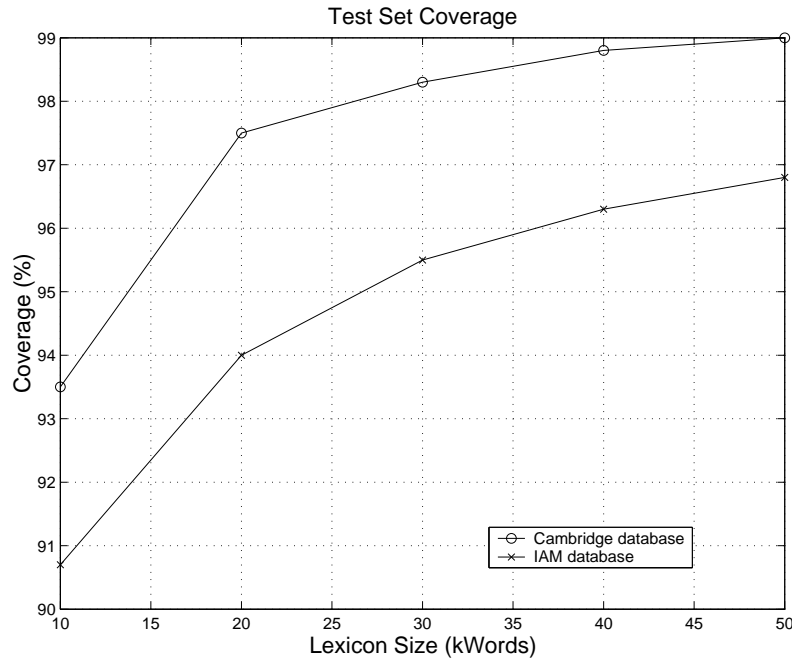
The next sections show how to select a lexicon and how the use of  $N$ -grams improves the performance of a recognition system.

### 12.6.1 Lexicon Selection

The lexicon of a recognizer is determined by the data to be recognized, e.g. in postal applications reading handwritten addresses the lexicon contains town names (see Section 12.8). In the case of unconstrained handwritten or spoken data, the only information available about the data to be recognized is the language that will be used. In fact, if the data are really unconstrained nothing can be said about their content. This is a problem because the recognizer can give as output only words belonging to the lexicon (see Section 12.4) and the number of OOV words must be minimized in order to reduce the amount of errors due to a mismatch between lexicon and data.

Although it is a generic information, the use of a language rather than another still represents an important constraint. In fact, this enables one to consider a corpus of texts written in a certain language and to extract from them the words most likely to appear in the data to be recognized. In the case of the experiments presented in this chapter, the language of the test data is English and the lexicon has been extracted from the TDT-2 corpus (see below for more details) [33], a collection of texts containing 20,407,827 words in total (the number of unique words appearing in the corpus is 192,209). A lexicon of size  $M$  can be obtained by extracting the  $M$  most frequent words in the corpus. The rationale behind such an approach is that the words appearing more frequently in a sufficiently representative collection of texts are likely to appear also in other texts written in the same language. This applies especially to *functional words* (prepositions, articles, conjunctions, etc.) and words of common use (*to be*, *to have*, etc.), but it is true also for other words.

The effectiveness of a lexicon can be measured in terms of *coverage*, i.e. percentage of the words in the test data that appear in the lexicon. The plot in Figure 12.7 shows the *coverage* with respect to the test set of the Cambridge and IAM databases for dictionaries of size ranging from 10,000 to 50,000. Although the text written in the test data is independent from the TDT-2



**Fig. 12.7.** Lexicon coverage. The plot shows the effect of the lexicon size on the coverage. 50,000 words are needed to cover more than 95% of the test data used in this chapter.

corpus, the 10,000 most frequent words appearing in this last cover more than 90% in both cases. On the other hand, the size of the lexicon must be multiplied by five to improve the coverage by few more points. This means that the increase of the coverage leads to undesirable effects such as heavier computational burden and highest misclassification probability. The effect of these two conflicting effects will be evident in the recognition results (see below) which show how the best WRR is achieved for lexicon sizes corresponding to a tradeoff between coverage improvement and above mentioned negative side-effects.

### 12.6.2 *N*-gram Model Performance

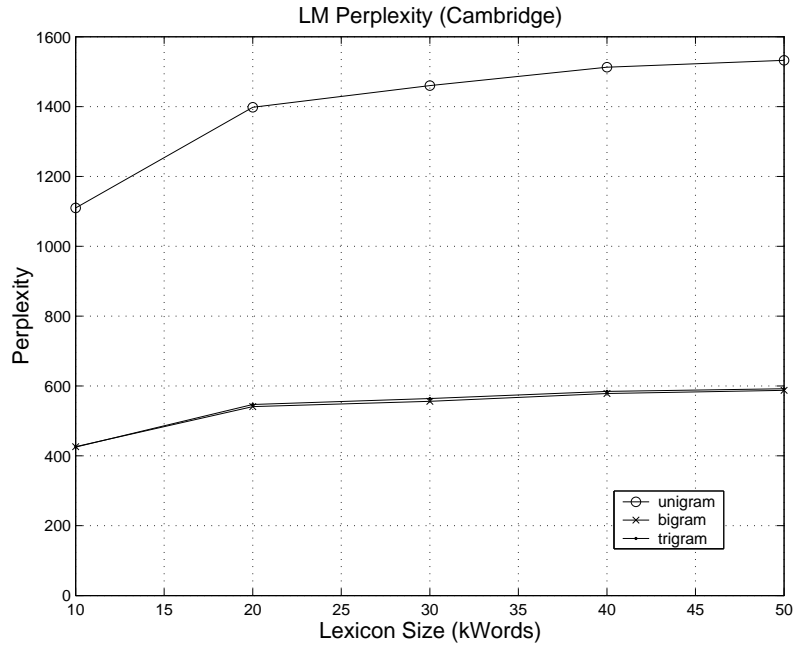
Once the lexicon has been selected, it is possible to train the *N*-gram models. In fact, the probabilities of observing the *N*-grams are estimated only for the words belonging to the lexicon. The reason is that these are the only words that exist for the recognizer. If  $M$  is the size of the lexicon, the number of possible *N*-grams is  $M^N$ , then the size of the language model increases quickly with the number of lexicon entries. This is one of the main reasons for keeping

as low as possible the size of the dictionary, while trying to increase as much as possible the coverage.

In the experiments presented in this chapter, the  $N$ -gram models have been trained over the TDT-2 corpus [33], a collection of transcriptions from several broadcast and newswire sources (ABC, CNN, NBC, MSNBC, Associated Press, New York Times, Voice of America, Public Radio International). For each one of the five lexica described above, three models (based on unigrams, bigrams and trigrams, respectively) have been created. The plots in Figure 12.8 and 12.9 show the perplexities of the  $N$ -grams as a function of the lexicon size. The perplexity is estimated over the part of the text covered by the lexicon, without taking into account OOV words. This happens because the only part of the text where the language model can be effective is the one covered by the lexicon. For this reason, we are interested to know the language model performance only over such part of the text. In practice, when an OOV word is encountered, the history is reset. The first term after the OOV word can be modeled only with unigrams, the second one at most with bigrams and only the third one can make use of trigrams. This increases significantly the perplexity and simulates the fact that the language model can only guess wrong words in correspondence of OOV words. The perplexity measured including the OOV words is lower, but less realistic with respect to the recognition task. In fact, during the recognition, the information about the presence of an OOV word is not available and the model can only guess a wrong word (independently of its perplexity). Some systems try to address the above limit by detecting the OOVs [40]. In this way, rather than giving a wrong transcription, the system provides a warning.

A significant improvement is obtained when passing from unigrams to bigrams, but no further improvement is obtained when applying trigrams. This happens for several reasons. The first is that the handwritten text is split into lines and only the words after the third one can take some advantages from the trigram model. Since a line contains on average 10 words, this means that only  $\sim 80\%$  of the data can actually benefit from the trigram model (while  $90\%$  of the data can be modeled with bigrams). A second problem is that the percentage of trigrams covered by the corpus in the test set is  $\sim 40\%$ . This further reduces the number of words where the trigram model can have a positive effect. The coverage in terms of bigrams is much higher (around  $85\%$ ) and the percentage of words over which the model can have an effect is more than  $90\%$ . On average, when trigrams are applied,  $\sim 45\%$  of the words in the test set are modeled with a trigram,  $\sim 40\%$  with a bigram and  $\sim 15\%$  with a unigram. This results in an average history length of 2.3. On the other hand, when the language is modeled with bigrams,  $\sim 85\%$  of the words are guessed with bigrams and  $\sim 15\%$  with unigrams. The resulting average history length is 1.8. For these reasons, the bigram and trigram models have a similar perplexity and do not make a big difference in terms of recognition performance.



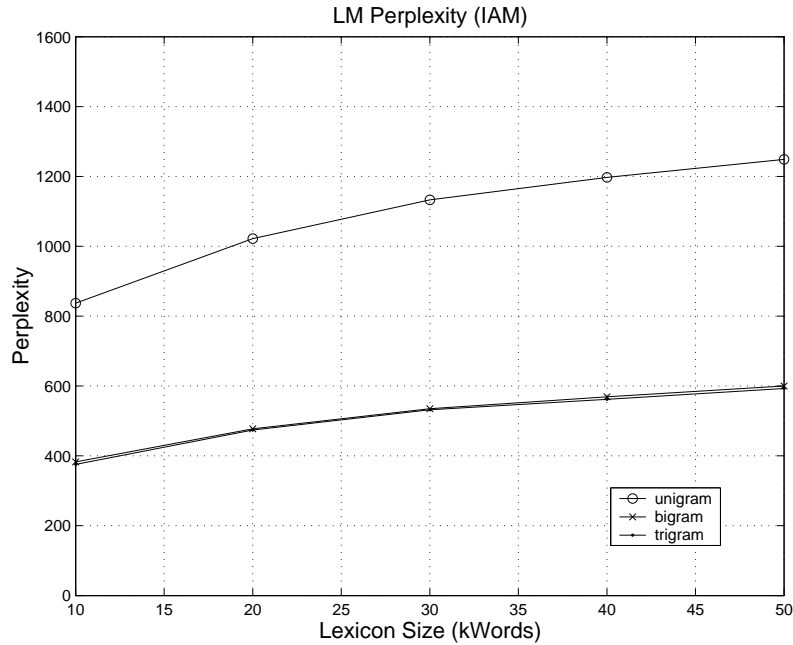


**Fig. 12.8.** Language model perplexity. The plots show the perplexity of the  $N$ -gram models as a function of the lexicon size.

In the case of spoken data, the effect due to the division into lines is not observed. On the other hand, other effects limit the effectiveness of the language models: the sentences boundaries are not detected, then the last words of a sentence and the first words of the following sentence form spurious  $N$ -grams. Moreover, hesitations and grammatical errors of the spontaneous speech are not modeled by  $N$ -grams trained on written, grammatically correct, texts.

### 12.6.3 Cambridge Database Results

This section reports the results obtained on the Cambridge database. Section 12.4 shows that the word models are obtained as concatenations of single letter models. In principle, the characteristics of each letter model should be set separately, but it is common practice to use the same topology for all letter models. In the case of handwriting, the topology is left-right (see Chapter 10) and the two parameters to set are the number of states  $S$  and the number of Gaussians  $G$  in the mixtures. The same value of  $S$  and  $G$  is used for every model and the results are satisfactory. Since  $S$  and  $G$  cannot be set a-priori, a validation phase is necessary. Models with  $10 \leq S \leq 14$  and  $10 \leq G \leq 15$  are trained over the training set and tested, without using  $N$ -grams, over the

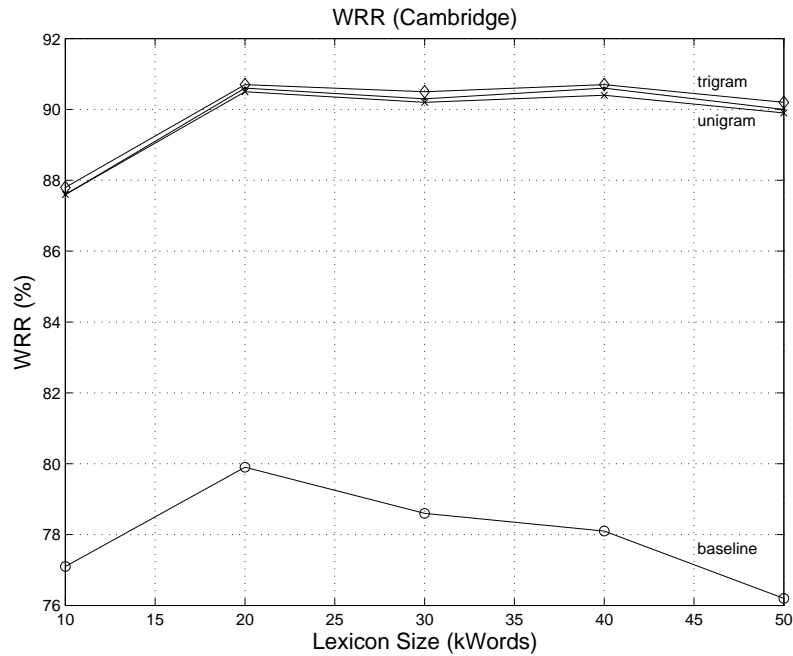


**Fig. 12.9.** Language model perplexity over the IAM test set. The plots show the perplexity of the  $N$ -gram models as a function of the lexicon size.

validation set. The system corresponding to the couple  $(S, G)$  giving the best results (over the validation set) is selected as optimal. The system selected in the validation phase ( $S = 12$  and  $G = 12$ ) is retrained over the union of training and validation set and the resulting system is used in the actual recognition experiments.

For each one of the five lexica described above, four versions of the system are tested over the test set. The first version (called *baseline*) makes no use of  $N$ -grams, the other ones use unigram, bigram and trigram models corresponding to the lexicon under consideration. The performance is measured using the WRR. The performance is the result of a tradeoff between the improvement of the test set coverage and the increase of the lexicon size. The application of the  $N$ -gram models has a significantly positive effect (the statistical confidence is higher than 90%). Moreover, the SLMs make the system more robust with respect to the increase of the lexicon size so that it is possible to maximize the benefit of the improved coverage.

The insertions have an important influence on the performance of the system. Sometimes, when part of a word corresponds to an entry in the lexicon (e.g. *unmentionable* is composed of the entries *un*, *mention* and *able*) the decoder favours the transcription splitting the bigger word, especially when the shorter words are more frequently represented in the training corpus. No

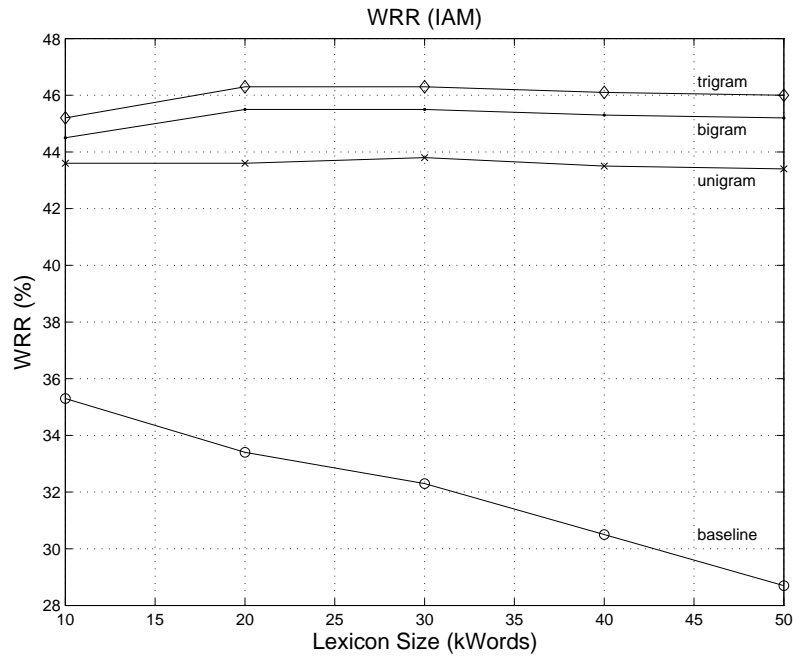


**Fig. 12.10.** System performance over Cambridge database. The plot on the left (right) shows the recognition rate (accuracy) of the system. The performance is measured over the test set for the four systems considered: baseline (no SLM), unigrams, bigrams and trigrams.

deletion error is observed. This is due to the fact that the spaces between neighboring words are typically evident, and are never missed (condition necessary to observe a deletion).

The systems using unigrams, bigrams and trigrams are equivalent in terms of performance. This is due, in our opinion, to the fact that the handwriting model alone has a high performance. The space for improvement is thus reduced. Most content words are recognized without the help of the language models.  $N$ -grams are actually helpful only to recognize functional words that are an important source of error because they are typically short (two or three letters). On the other hand, the performance of the language models over the functional words is not significantly improved by increasing their order. For this reason, the use of bigrams and trigrams does not result in a higher recognition or accuracy.

The situation is different for multiple writer (or speaker) data where the handwriting (or acoustic) model alone is weak. In this case, the HMMs have a low performance over the words where  $N$ -grams of different order have a sig-



**Fig. 12.11.** System performance over IAM database. The plot shows the word recognition rate over the test set of the IAM database.

nificantly different effectiveness. This leads to an improvement when passing from unigrams to trigrams.

#### 12.6.4 IAM Database Results

This section describes the results obtained over the IAM database. The parameters  $S$  and  $G$  were set using the same method as described in the previous section for Cambridge database. Models with  $19 \leq S \leq 23$  and  $10 \leq G \leq 15$  are trained over the training set and tested, without using  $N$ -grams, over the validation set. The selected model ( $S = 20$  and  $G = 12$ ) is retrained over the union of training and validation set and it is used in the actual recognition experiments.

The dictionaries and the language models are the same as those used in the single writer experiments. The performance of the systems is measured in terms of WRR (see previous section). For each dictionary, four recognizers are tested: the first (called *baseline*) makes no use of language models. The others use alternatively unigrams, bigrams and trigrams.

Also in this case, the use of  $N$ -grams has a two-fold positive effect: the performance is not only improved (independently of the metric used), but the

system is also more robust with respect to an increase of the lexicon size. Figure 12.11 shows that the performance of the systems using the language models is stable when the lexicon size passes from 10,000 to 50,000, while accuracy and recognition of the baseline system are significantly lowered.

The increase of the language model order produces an improvement (statistical significance higher than 90%). The language models can play a role not only over the functional words (see previous section) but also over the content words where the difference of the order results in a better local perplexity. The error is mostly due to substitution (around 45%). Insertion and deletion rates are about 9% and 4%, respectively.

## 12.7 Speech Recognition Results

This section presents the results achieved with state-of-the-art recognition systems over different kinds of data, namely broadcast news, phone conversational speech and meeting recordings. The literature presents results achieved over many other sources of data, but the three above are the most representative and challenging for nowadays speech recognition systems. Moreover, the most important international evaluations are performed using exactly such kind of data.

The automatic transcription of broadcast data is one of the most investigated tasks in speech recognition. This has two main reasons: the first is that the news are an important source of information and their transcription has important applications, the second is that broadcast news data are particularly suitable for speech recognition. In fact, news are typically read by professional speakers in a radio or television studio where there are no background noises or other disturbing phenomena. Moreover, news speakers typically read a text that respects grammatical rules. As a consequence, the language models trained over text corpora fit well the content of the spoken data and no disfluencies like interruptions, repetitions or hesitations are observed.

For the above reasons, speech recognition performances obtained over broadcast news are typically higher than those obtained over other kinds of data. Table 12.1 shows the results obtained in one of the last international evaluations carried out by the *National Institute for Standards and Technology* (NIST). The tests are performed over 6 hours of news with a dictionary of 59,000 words and more complete results are reported in [26][62]. The table shows that the average WER is around 13% for all systems, but for some specific speakers the error rate goes down to around 5%. All systems are based on the approach described in the previous part of the chapter, i.e. they use hidden Markov models for the acoustic modeling and apply 3-grams as language models.

The NIST organizes an evaluation also for phone conversational speech. This is a rather difficult task because it combines two major sources of dif-

**Table 12.1.** Speech recognition results on broadcast news. The table reports the results obtained over broadcast news by several groups: LIMSI (at INRIA in France), BBN (Boston, USA), Cambridge University (CU) and a combination of LIMSI and CU systems.

| System   | WER (%) |
|----------|---------|
| LIMSI    | 12.8    |
| BBN      | 13.0    |
| CU       | 13.3    |
| LIMSI/CU | 13.0    |

**Table 12.2.** Speech recognition results on phone conversational speech. The table reports the results obtained over phone conversations by SRI+ICSI+UW and LIMSI+BBN.

| System      | WER (%) |
|-------------|---------|
| LIMSI+BBN   | 21.5    |
| SRI+ICSI+UW | 24.0    |

difficulties: the first is the low quality of phone speech due to noise and low sampling frequency (8 kHz). The second is that conversational speech includes phenomena hard to tackle such as disfluencies (see above) and overlapping speech. Table 12.2 shows the results obtained over 6 hours of phone conversations using a dictionary of 59000 words by two groups: the first includes Stanford Research Institute (SRI), International Computer Science Institute (ICSI) and University of Washington (UW) [83], the second includes the LIMSI and BBN [62].

Meeting data are more and more frequently used because they enable researchers to investigate new kinds of problems, namely the high number of speakers involved in a given recording, a wide range of dialogue phenomena (interruptions, floor grabbing, etc.), and the so-called back-channel interjections, i.e. expressions like *yeah* or *hmm* that overlap the utterances of the person holding the floor at a given time. Descriptions of experiments and results obtained by different groups over the same data are presented in [35] and the WER is around 30%.

## 12.8 Applications

This section shows the major applications involving speech and handwriting recognition technologies. Although both domains are still subject of research and the recognition problem cannot be considered solved, there are some real world applications where tight experimental constraints (see below for more details) make the recognition easy enough to achieve satisfactory results. The

next two sections show some of the more successful cases for both speech and handwriting.

### 12.8.1 Applications of Handwriting Recognition

Many works about handwriting recognition are dedicated to the bank check legal amount recognition. The developed systems are good enough to be used in commercial products as described in [32], where a family of systems able to work on french, english and american checks is claimed to have a performance close to a human reader (rejecting 30-40% of the data).

The reading of legal amounts involves small lexicons (between 25 and 30 words) and each word to be recognized is produced by a different writer. An important advantage in bank check reading is the presence of the courtesy amount (the amount written in digits). This can be read reliably, but is not relevant from a legal point of view, so an automatic check processing system must read also the amount written in letters. On the other hand, the redundancy of information when reading both courtesy and legal amount can improve the performance of the system.

In [47][65][67], an implicit segmentation is applied, and the recognition is performed with an Hidden Markov Model. A segmentation free approach is proposed in [34][57][91], where a sliding window is applied to the amount image and a recurrent neural network is used in conjunction with HMMs. The sliding window is also used in [43], where a correction mechanism activated by a mismatch between courtesy and legal amount is proposed. In [73], the scaled images of the legal amount are considered as random field realizations and recognized in conjunction by HMM and Markov random fields [15]. A combination of methods based on analytic and global features was presented in [17][18][19][20]. This approach is especially conceived to work on italian amounts: these are more difficult to recognize because they are obtained by joining several basic words. In [32], the human performance is said to be around 99%. This rate can be achieved by current automatic readers only by discarding the more ambiguous samples.

Although the success of bankcheck reading systems, most works in the literature concern postal applications. The data involved in this domain are completely unconstrained, each word is written by a different writer, the words can be cursive, handprinted or a mix of the two styles. The lexicon depends, in general, on the output of a zip code recognizer. When the zip code is recognized, it is not even necessary to read further informations in the address. When there is unacceptable ambiguity on the last, last two or last three digits of the zip code, then it is necessary to read the town name and the lexicon will contain ten, hundred or towсанд words respectively.

Several works are based on segmentation and dynamic programming [13][24][63][77]. In [63], the performance is improved by using, together with the segmentation based system, a segmentation free system based on HMM. The combination of two different approaches (lexicon free and lexicon directed) is

also described in [77][81]. Techniques to calculate the score of a lexicon word, given the single character confidences, are proposed in [13] and [24].

A system based on HMM is presented in [12], where a modified Viterbi algorithm is described. In [23], after having performed an explicit segmentation, the system uses an HMM based technique to combine two feature sets: the first oriented to characters, the second to ligatures. The segmentation statistics (the probability of segmenting each letter into  $n$  primitives) are taken into account during the recognition process in [46][11][45][49]. A minimum edit distance modified to better represent the errors that can occur in a cursive word recognition is described in [74]. In [22][55][54][68], the possibility of reading handwritten lines is investigated to recognize different forms assumed by the same address (e.g. *Park Avenue* or *Park Av.*).

The current frontier in handwriting recognition is the automatic transcription of unconstrained documents, where the handwritten information is mixed with other kinds of data and where there is no hint about the content (apart the language of the text). In postal and bankcheck applications, the environment involving the system is a source of informations that have a strong influence on the recognition process. In the works presented in this section, the recognition was performed over data that did not allow the use of any other information than the handwritten words themselves. At most, if the words belong to a text, the linguistic knowledge would be introduced. The data used in the works related to this subfield of cursive word recognition is often created ad hoc by asking writers (in some case cooperatives) to produce samples.

In [6][21] the words produced by few writers are recognized. Both works are based on explicit segmentation and use different level representations of the words that allow making hypotheses about the transcription and looking for its confirmation at the feature level. In [21], the confirmation is obtained as a degree of alignment of letter prototypes with the actual handwritten data. In [6] the confirmation is given by matching the sequence of expected events (e.g. loops, curve strokes of various shape, etc.) with the actual sequence detected in the handwritten word.

In [92][93], a word is segmented explicitly first and then an HMM is used to find the best match between the fragments (the similarity with character prototypes is used to calculate the probability of a fragment being a certain letter) and words in the lexicon. In [7][8], the words written by cooperative writers are represented, after a skeletonization of the word, as a sequence of strokes organized in a graph. Each stroke is represented by a feature vector and their sequence is recognized by an HMM.

The first example of recognition of data extracted from a text (to our knowledge) is presented in [75][76]. The selection of the text is addressed by linguistic criteria, the text is extracted from a corpus supposed to be representative of the current English. This allows the use of linguistic knowledge in recognition. The data is produced by a single writer, so that an adaptation to his/her writing style can play a role in improving the system performance.



In [75][76], the words are skeletonized and then a uniform framing is performed. From each frame a feature vector is extracted and an HMM is used for the recognition. A recurrent neural network is used to calculate the emission probabilities of the HMM.

The use of linguistic knowledge was shown to be effective in recognizing whole sentences rather than single words in [56][58][59][60][89][90][96]. The applied language models are based on statistic distributions of unigrams and bigrams [40]. The use of syntactical constraints (expressed by transition probabilities between different syntactical categories) was experimented in [79][80].

### 12.8.2 Applications of Speech Recognition

One of the most successful applications of speech recognition is the automation of *customer care systems*, i.e. phone based business services addressing the needs of clients calling by phone. Following [2], such systems perform three tasks of increasing complexity: the first is the recognition of commands belonging to a predefined set (often small), the second is the so-called *call routing*, i.e. the redirection of the call to an appropriate operator based on the needs expressed by the clients, and the third is the *information gathering*, i.e. the extraction of specific data (e.g. addresses or names) from client calls.

The recognition of commands involves two strong constraints, the small dictionary size and the command grammar, i.e. the rules that must be respected for command sequences being valid. The application of such a technology to a *smart home*, i.e. an apartment where the devices can be activated through vocal commands, has been investigated in [64] and the results show that the major problem is the noise due to both environment and communication channels (e.g. phones) used to send input to the systems. The same applies to the recognition of commands for cellular phones [16][87] as well as portable devices [10][53], and the approach commonly applied is to make the front end more robust with respect to the noise. In general, the command recognition systems associated to small devices, must be trained by the user by repeating a certain number of times the commands to be transcribed. Although such an effort is small, still it represents an obstacle for many users and this has limited the diffusion of this kind of products.

The second task by increasing complexity order (see above) is the call routing. The first real-world system was deployed in the nineties [31] and it was based on the selection of *salient* segments from the automatic transcription of calls. However, the attempt to understand the calls has been quickly abandoned in favor of Information Retrieval (see below) oriented approaches, i.e. of techniques that model texts as vectors and apply pattern recognition techniques in order to correctly classify the call [14] and such an approach is still today dominant [38][50][52]. The main problem of this approach is that it requires large amounts of labeled material which is not always easy to obtain and some attempts are currently being made in order to build effective routers with little data [86].

The third problem, i.e. the information gathering, has been addressed only recently and it is still at an early stage due to its multiple difficulties. A recent approach is the so-called *distillation* [29][70][71], i.e. the use of templates like *The player XXX has scored in the YYY vs ZZZ match* to find data segments likely to provide important information. The main problem of such a technique is that it can be difficult to find templates precise enough to capture only relevant information and flexible enough to cover all variants of the same statement. Different approaches include the use of submitted queries as templates [9][10][37][97], the identification of the topic [51], the detection of *discourse markers* [44] or more application specific criteria [66].

The second important domain of application of speech recognition system is the *spoken document retrieval*, i.e. the application of information retrieval technologies to automatic transcription of speech recordings. Most of the research on the retrieval of speech recordings has been made in the framework of the TREC conferences<sup>5</sup> [27]: several groups worked on the same database (TDT-3 [33]) containing both manual (WRR ~90 percent) and automatic (WRR ~70 percent) transcriptions of broadcast news recordings. The TDT-3 dataset is composed of around 25,000 documents and in addition a set of queries with their respective relevance judgements. The participants equipped with an ASR system could use their own transcriptions which enabled the evaluation of the WER impact on the retrieval performance. The works presented in the TREC context do not try to model the noise: the techniques successfully applied on clean texts have been shown to be effective also on noisy automatic transcriptions. All systems are based on the Vector Space Model (VSM) [3], where documents and queries are converted into vectors and then compared through matching functions. In most cases, the documents are indexed with *tf.idf* [3] and matched with the Okapi formula [1][25][28][41], along with other approaches [36][48][78]. During the extensive experiments and comparisons performed in the TREC framework, at least two important conclusions emerge: (a) The retrieval is more effective over transcriptions at the word, rather than at the phoneme level. Some attempts were made to recognize documents as phoneme sequences and then to match them with the query words, but the performances were much lower than in the alternative approach [27]. (b) There is almost no retrieval performance degradation when increasing the WER from around 10 percent to around 40 percent [27].

---

<sup>5</sup> At the time this book is being written, the proceedings are available online at the site <http://nist.trec.gov>.

---

## References

1. D. Abberley, S. Renals, D. Ellis, and T. Robinson. The THISL SDR system at TREC-8. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 699–706, 1999.
2. D. Attwater, M. Edgington, P. Durston, and S. Whittaker. Practical issues in the application of speech technology to network and customer service applications. *Speech Communication*, 31(4):279–291, 2000.
3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
4. L.R. Bahl, V. De Gennaro, P.S. Gopalakrishnan, and R.L. Mercer. A fast approximate acoustic match for large vocabulary speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(1):59–67, 1993.
5. H. Bourlard and N. Morgan. *Connectionist Speech Recognition - A Hybrid Approach*. Kluwer, 1994.
6. R.M. Bozinovic and S.N. Srihari. Off-line cursive script word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):69–83, January 1989.
7. H. Bunke, M. Roth, and E.G. Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden Markov models. *Pattern Recognition*, 28(9):1399–1413, September 1995.
8. Horst Bunke, M. Roth, and E.G. Schukat-Talamazzini. Off-line recognition of cursive script produced by a cooperative writer. In *Proceedings of International Conference on Pattern Recognition*, pages 383–386, 1994.
9. W. Byrne, D. Doermann, M. Franz, S. Gustman, J. Hajic, D. Oard, M. Picheny, J. Psutka, B. Ramabhadran, D. Soergel, T. Ward, and Wei-Jing Zhu. Automatic recognition of spontaneous speech for access to multilingual oral history archives. *IEEE Transactions on Speech and Audio Processing*, 12(4):420–435, 2004.
10. E. Chang, F. Seide, H.M. Meng, Zhuoran Chen, Yu Shi, and Yuk-Chi Li. A system for spoken query information retrieval on mobile devices. *IEEE Transactions on Speech and Audio Processing*, 10(8):531–541, 2002.
11. M.Y. Chen and A. Kundu. An alternative to variable duration HMM in handwritten word recognition. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition*, 1993.
12. M.Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition using a hidden Markov model type stochastic network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):481–496, May 1994.

13. W. Chen, P. Gader, and H. Shi. Lexicon-driven handwritten word recognition using optimal linear combinations of order statistics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(1):77–82, January 1999.
14. J. Chu-Carroll and B. Carpenter. Vector based natural language call routing. *Computational Linguistics*, 25(3):361–388, 1999.
15. F.S. Cohen. Markov random fields for image modelling e analysis. In U. Desai, editor, *Modelling and Applications of Stochastic Processes*, pages 243–272. Kluwer Academic Press, 1986.
16. S. Deligne, S. Dharanipragada, R. Gopinath, B. Maison, P. Olsen, and H. Printz. A robust high accuracy speech recognition system for mobile applications. *IEEE Transactions on Speech and Audio Processing*, 10(8):551–561, 2002.
17. V. Di Lecce, A. Dimauro, Guerriero, S. Impedovo, G. Pirlo, and A. Salzo. A new hybrid approach for legal amount recognition. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition*, pages 199–208, Amsterdam, 2000.
18. G. Dimauro, S. Impedovo, and G. Pirlo. Automatic recognition of cursive amounts on italian bank-checks. In S. Impedovo, editor, *Progress in Image Analysis and Processing III*, pages 323–330. World Scientific, 1994.
19. G. Dimauro, S. Impedovo, G. Pirlo, and A. Salzo. Bankcheck recognition systems: re-engineering the design process. In A. Downton and S. Impedovo, editors, *Progress in Handwriting Recognition*, pages 419–425.
20. G. Dimauro, S. Impedovo, G. Pirlo, and A. Salzo. Automatic bankcheck processing: A new engineered system. In *Automatic Bankcheck Processing*, pages 5–42. World Scientific Publishing, 1997.
21. S. Edelman, T. Flash, and S. Ullman. Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, 5(3):303–331, March 1990.
22. A. El Yacoubi, J.M. Bertille, and Gilloux M. Conjoined location and recognition of street names within a postal address delivery line. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 1024–1027, Montreal, 1995.
23. A. El-Yacoubi, M. Gilloux, R. Sabourin, and C.Y. Suen. An HMM,-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):752–760, August 1999.
24. John T. Favata. General word recognition using approximate segment-string matching. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 92–96, Ulm, 1997.
25. M. Franz, J.S. McCarley, and R.T. Ward. Ad hoc, cross-language and spoken document information retrieval at IBM. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 391–398, 1999.
26. M.J. Gales, D.Y. Kim, P.C. Woodland, H.Y. Chan, D. Mrva, R. Sinha, and S.A. Tranter. Progress in the CU-HTK boradcast news transcription system. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1513–1525, 2006.
27. J.S. Garofolo, C.G.P. Auzanne, and E.M. Voorhees. The TREC spoken document retrieval track: A success story. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 107–129, 1999.
28. J.L. Gauvain, Y. de Kercadio, L. Lamel, and G. Adda. The LIMSI SDR system for TREC-8. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 475–482, 1999.

29. C. Gerber. Found in translation. *Military Information Technology*, 10(2), 2006.
30. P.S. Gopalakrishnan, L.R. Bahl, and R.L. Mercer. A tree search strategy for large vocabulary continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustic, Speech and Signal Processing*, pages 572–575, 1995.
31. A. Gorin, G. Riccardi, and J. Wright. How may I help you? *Speech Communication*, 23(2):113–127, 1997.
32. N. Gorski, V. Anisimov, E. Augustin, O. Baret, D. Price, and J.C. Simon. A2iA check reader: A family of bank check recognition systems. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 523–526, Bangalore, 1999.
33. D. Graff, C. Cieri, S. Strassel, and N. Martey. The TDT-3 text and speech corpus. In *Proceedings of Topic Detection and Tracking Workshop*, 2000.
34. D. Guillevic and C.Y. Suen. HMM word engine recognition. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 2, pages 544–547, Ulm, 1997.
35. T. Hain, L. Burget, J. Dines, G. Garau, M. Karafiat, M. Lincoln, J. Vepa, and V. Wan. The AMI meeting transcription system: progress and performance. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2007.
36. B. Han, R. Nagarajan, R. Srihari, and M. Srikanth. TREC-8 experiments at SUNY at Buffalo. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 591–596, 1999.
37. J.H.L. Hansen, R. Huang, B. Zhou, M. Seadle, J.R. Deller, A.R. Gurijala, M. Kurimo, and P. Angkititrakul. Speechfind: Advances in spoken document retrieval for a national gallery of the spoken word. *IEEE Transactions on Speech and Audio Processing*, 13(5):712–730, 2005.
38. Q. Huang and S. Cox. Task-independent call-routing. *Speech Communication*, 48(3-4):374–389, 2006.
39. X. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice-Hall, 2001.
40. F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.
41. S.E. Johnson, P. Jourlin, K. Spärck-Jones, and P.C. Woodland. Spoken document retrieval for TREC-8 at Cambridge University. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 197–206, 1999.
42. D. Jurafsky and J.H. Martin. *Speech and Language Processing: an Introduction to Natural Processing Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.
43. G. Kaufmann and H. Bunke. Automated reading of cheque amounts. *Pattern Analysis and Applications*, 3:132–141, march 2000.
44. T. Kawahara, M. Hasegawa, K. Shitaoka, T. Kitade, and H. Nanjo. Automatic indexing of lecture presentations using unsupervised learning of presumed discourse markers. *IEEE Transactions on Speech and Audio Processing*, 12(4):409–419, 2004.
45. G. Kim and V. Govindaraju. Handwritten word recognition for real time applications. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 24–27, Montreal, 1995.
46. G. Kim and V. Govindaraju. A lexicon driven approach to handwritten word recognition for real time application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):366–379, 1997.

47. S. Knerr, E. Augustin, O. Baret, and D. Price. Hidden Markov model based word recognition and its application to legal amount reading on French checks. *Computer Vision and Image Understanding*, 70(3):404–419, June 1998.
48. W. Kraaij, R. Pohlmann, and D. Hiemstra. Twenty-one at TREC-8 using language technology for information retrieval. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 285–300, 1999.
49. A. Kundu, Y. He, and M.Y. Che. Alternatives to variable duration HMM in handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1275–1280, November 1998.
50. H.-K.J. Kuo and L. Chin-Hui. Discriminative training of natural language call routers. *IEEE Transactions on Speech and Audio Processing*, 11(1):24–35, 2003.
51. M. Kurimo. Thematic indexing of spoken documents by using self-organizing maps. *Speech Communication*, 38(1-2):29–45, 2002.
52. C.H. Lee, B. Carpenter, W. Chou, J. Chu-Carroll, W. Reichl, A. Saad, and Q. Zhou. On natural language call routing. *Speech Communication*, 31(4):309–320, 2000.
53. D. Li, W. Kuansan, A. Acero, H. Hsiao-Wuen, J. Droppo, C. Boulis, W. Ye-Yi, D. Jacoby, M. Mahajan, C. Chelba, and X.D. Huang. Distributed speech processing in miPad’s multimodal user interface. *IEEE Transactions on Speech and Audio Processing*, 10(8):605–619, 2002.
54. S. Madhvanath, E. Kleinberg, and V. Govindaraju. Holistic verification of handwritten phrases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999.
55. S. Madhvanath, E. Kleinberg, V. Govindaraju, and S.N. Srihari. The HOVER system for rapid holistic verification of off-line handwritten phrases. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 2, pages 855–859, Ulm, 1997.
56. U. Marti and H. Bunke. Towards general cursive script recognition. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition*, pages 379–388, Korea, 1998.
57. U. Marti, G. Kaufmann, and Bunke H. Cursive script recognition with time delay neural networks using learning hints. In W. Gerstner, A. Gernoud, M. Hasler, and J.D. Nicoud, editors, *Artificial Neural Networks - ICANN97*, pages 973–979. Springer Verlag, 1997.
58. U.-V. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 705–708, Bangalore, 1999.
59. U.V. Marti and H. Bunke. Handwritten sentence recognition. In *Proceedings of International Conference on Pattern Recognition*, volume 3, pages 467–470, Barcelona, 2000.
60. U.V. Marti and H. Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence*, 2001.
61. U.V. Marti and H. Bunke. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal of Document Analysis and Recognition*, 5(1):39–46, january 2002.
62. S. Matsoukas, J.L. Gauvain, G. Adda, T. Colthurst, C.L. Kao, O. Kimball, L. Lamel, F. Lefevre, J.Z. Ma, J. Makhoul, L. Nguyen, R. Prasad, R. Schwartz, H. Schwenk, and B. Xiang. Advances in transcription of broadcast news and

- conversational telephone speech within the combined EARS BBN/LIMSI. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1541–1556, 2006.
63. M. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):548–554, May 1996.
  64. S. Möller, J. Krebber, and P. Smeele. Evaluating the speech output component of a smart-home system. *Speech Communication*, 48(1):1–27, 2006.
  65. C. Olivier, T. Paquet, M. Avila, and Y. Lecourtier. Recognition of handwritten words using stochastic models. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 19–23, Montreal, 1995.
  66. M. Padmanabhan, G. Saon, J. Huang, B. Kingsbury, and L. Mangu. Automatic speech recognition performance on a voicemail transcription task. *IEEE Transactions on Speech and Audio Processing*, 10(7):433–442, 2002.
  67. T. Paquet and Y. Lecourtier. Recognition of handwritten sentences using a restricted lexicon. *Pattern Recognition*, 26(3):391–407, 1993.
  68. J. Park, V. Govindaraju, and S.N. Srihari. Efficient word segmentation driven by unconstrained handwritten phrase recognition. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 1, pages 605–608, Bangalore, 1999.
  69. R. Plamondon and S.N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.
  70. D. Poncelon and S. Srinivasan. Automatic discovery of salient segments in imperfect speech transcripts. In *ACM Conference on Information and Knowledge Management*, pages 490–497, 2001.
  71. D. Poncelon and S. Srinivasan. Structure and content based segmentation of speech transcripts. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 404–405, 2001.
  72. L.R. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.
  73. G. Saon. Cursive word recognition using a random field based hidden Markov model. *International Journal of Document Analysis and Recognition*, 1(1):199–208, 1999.
  74. G. Seni, V. Kripasundar, and R.K. Srihari. Generalizing edit distance to incorporate domain information: Handwritten text recognition as a case study. *Pattern Recognition*, 29(3):405–414, 1996.
  75. A.W. Senior. *Off-Line Cursive Handwriting Recognition Using Recurrent Neural Network*. PhD thesis, University of Cambridge, UK, 1994.
  76. A.W. Senior and A.J. Robinson. An off-line cursive handwriting recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):309–321, March 1998.
  77. M. Shridar, G. Houle, and Kimura F. Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 2, pages 861–865, Ulm, 1997.
  78. A. Singhal, S. Abney, M. Bacchiani, M. Collins, D. Hindle, and F. Pereira. AT&T at TREC-8. In *Proceedings of 8<sup>th</sup> Text Retrieval Conference*, pages 317–330, 1999.

79. R.K. Srihari. Use of lexical and syntactic techniques in recognizing handwritten text. In *Proceedings of ARPA workshop on Human Language Technology*, pages 403–407, 1994.
80. R.K. Srihari and C. Baltus. Incorporating syntactic constraints in recognizing handwritten sentences. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1262–1267, 1993.
81. S.N. Srihari. Handwritten address interpretation: a task of many pattern recognition problems. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(5):663–674, 2000.
82. T. Steinherz, E. Rivlin, and N. Intrator. Off-line cursive script word recognition - a survey. *International Journal on Document Analysis and Recognition*, 2(2):1–33, 1999.
83. A. Stolcke, B. Chen, H. Franco, V.R. Rao Gadde, M. Graciarena, M.Y. Hwang, K. Kirchhoff, A. Mandal, N. Morgan, X. Lei, T. Ng, M. Ostendorf, K. Sönmez, A. Venkataraman, D. Vergyri, W. Wang, J. Zheng, and Q. Zhu. Recent innovations in speech-to-text transcriptions at SRI-ICSI-UW. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1729–1744, 2006.
84. Lee S.W., editor. *Advances in Handwriting Recognition*. World Scientific Publishing Company, 1999.
85. O.D. Trier, A.K. Jain, and T. Taxt. Feature extraction methods for character recognition—A survey. *Pattern Recognition*, 10(4):641–662, 1996.
86. G. Tur, R. Schapire, and D. Hakkani-Tr. Active learning for spoken language understanding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2003.
87. I. Varga, S. Aalborg, B. Andrassy, S. Astrov, J.G. Bauer, C. Beaugeant, C. Geissler, and H. Hoge. ASR in mobile phones - an industrial approach. *IEEE Transactions on Speech and Audio Processing*, 10(8):562–569, 2002.
88. A. Vinciarelli. A survey on off-line cursive word recognition. *Pattern Recognition*, 35(7):1433–1446, 2002.
89. A. Vinciarelli. Noisy text categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1882–1895, 2005.
90. A. Vinciarelli, S. Bengio, and H. Bunke. Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):709–720, 2004.
91. W. Wang, A. Brakensiek, A. Kosmala, and G. Rigoll. HMM based high accuracy off-line cursive handwriting recognition by a baseline detection error tolerant feature extraction approach. In *Proceedings of International Workshop on Frontiers in Handwriting Recognition*, pages 209–218, Amsterdam, 2000.
92. B.A. Yanikoglu and P.A. Sandon. Off line cursive handwriting recognition using neural networks. In *Proceedings of SPIE Conference on Applications of Artificial Neural Networks*, 1993.
93. B.A. Yanikoglu and P.A. Sandon. Off-line cursive handwriting recognition using style parameters. *Tech. Rep. PCS-TR93-192 Dartmouth College*, 1993.
94. S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK book*. <http://htk.eng.cam.ac.uk/docs/docs/shtml>, 2000.
95. M. Zimmermann and H. Bunke. Automatic segmentation of the IAM off-line database for handwritten english text. In *Proceedings of 16<sup>th</sup> International Conference on Pattern Recognition*, volume IV, pages 35–39, 2002.



96. M. Zimmermann, J.-C. Chappelier, and H. Bunke. Offline grammar-based recognition of handwritten sentences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):818–821, 2006.
97. V. Zue, S. Seneff, J.R. Glass, J. Polifroni, C. Pao, T.J. Hazen, and L. Hetherington. Juplter: a telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85–96, 2000.



## Automatic Face Recognition

### What the reader should know to understand this chapter

- Basic notions of image processing (Chapter 3).
- Support vectors machines and kernel methods (Chapter 9).
- Principal component analysis (Chapter 11).

### What the reader should know after reading this chapter

- State-of-the-art in automatic face recognition.
- How to implement a basic automatic face recognition system.
- How to measure the performance of a face recognition system.

### 13.1 Introduction

The problem of automatic face recognition (AFR) can be stated as follows: given an image or a video showing one or more persons, recognize the individuals that are portrayed in a predefined dataset of face images [72]. Such a task has been studied for several decades. The earliest works appeared at the beginning of the 1970s [28][29], but it is only in the last few years that the domain has reached its maturity. The reason is twofold: on one hand, necessary computational resources are now easily available and recognition approaches achieve, at least in controlled conditions, satisfactory results. On the other hand, several applications of commercial interest require robust face recognition systems, e.g. multimedia indexing, tracking, human computer interaction, etc.

However, the most common and important application of face recognition is *secure access*, i.e. the control of the access to electronic resources such as bank accounts, confidential documents, sites requiring parental control, etc. In other words, the face is supposed to replace, or at least to support, the most common form of secure access, i.e. the entry of a *password*. From this

point of view, face recognition can be considered as a branch of *biometry*, the domain that tries to use physical human characteristics, e.g. fingerprint, voice, etc., to certify the identity of a person. The advantages of biometry over password entry are multiple: physical characteristics are more difficult to forge or steal, they cannot be *forgotten* as can happen with a password; they are virtually different for each human being, etc. Some biometric features, especially fingerprints, are more effective than faces as a recognition clue, but the use of the face has some important advantages [49]; in particular, people can be identified without their collaboration, i.e. without explicitly participating in an identification process. This is especially important in analyzing media (the people appearing in news or movies are not necessarily available), remote surveillance applications, detection of suspects in an open environment, etc.

The first part of this chapter provides a general overview of the AFR domain and describes the different approaches presented in the literature (see [72] for an extensive survey). The major steps of the recognition process are shown in detail and, for each one of them, a survey of the most important methods is proposed. While Chapter 12 has shown that the same machine learning approach can be applied to different kinds of data, this chapter shows that different machine learning approaches are applied to the same data. In fact, AFR can be considered as a classification problem and most of the classifiers presented in the previous part of the book (neural networks, support vector machines, etc.) have been applied in face recognition.

The second part of the chapter presents experiments performed with a system often used as a baseline for comparison with more sophisticated approaches. The system extracts PCA features from the face images and classifies them using support vector machines (see the rest of the chapter for more details). The goal of the experiments is not to show state-of-the-art results, more recent techniques achieve better performances, but rather to provide practical details for laboratory exercises on an AFR toy problem. In fact, the experiments are performed with software and data that can be obtained on the web: the XM2VTS face database, the *TorchVision* utilities<sup>1</sup> for face recognition and the *svmLight* SVM package (see Chapter 9). Special attention will be paid to *TorchVision*, a user-friendly package implementing all the steps of the face recognition process. Detailed descriptions of the *TorchVision* functions are provided throughout the chapter.

Face recognition technologies and, more generally, biometry are the subject of both surveys [23][71][72] and monographies [24][32][66] that the interested reader can consult for deeper information. Recent approaches involve the recognition of 3D face models [33][57], but this chapter focuses on the recognition of face images, a domain that is better established and has been extensively investigated in the literature.

---

<sup>1</sup> At the time this book is being written, the *TorchVision* utilities and the features extracted from the XM2VTS dataset are available on the following website: <http://pyverif.idiap.ch>.

The rest of this chapter is organized as follows: Section 13.2 describes the general architecture of an AFR system. Section 13.3 presents the face localization problem. Section 13.4 introduces the face image normalization techniques. Section 13.5 presents the most common feature extraction techniques used in AFR. Section 13.6 describes the most common machine learning approaches used in AFR. Section 13.7 shows the most important databases available for benchmarking purposes. Section 13.8 presents the laboratory experiments.

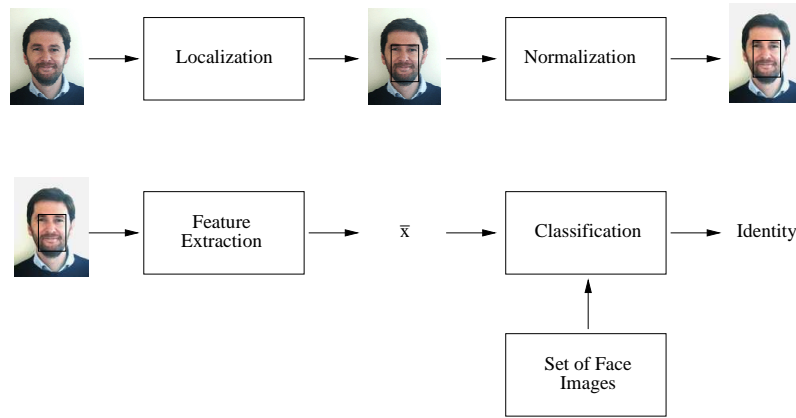
## 13.2 Face Recognition: General Approach

The general architecture of a face recognition system is shown in Figure 13.1. The recognition process includes four main steps: *localization*, *normalization*, *feature extraction* and *recognition*.

The first step of the process is the identification of the correct position of the face in an input image. The localization can be considered as a simplified version of the more general *detection* problem, i.e. of the localization of an arbitrary number of faces appearing in arbitrary positions in an image (see Section 13.3). In the case of the AFR systems, the images are supposed to contain only one face and the images are typically centered around the face under examination. However, even in such a simplified situation, localization errors can still happen and have an influence on the recognition results [41][54].

The normalization, like in the case of handwriting and speech recognition (see Chapter 12), is the elimination of the variability unnecessary to the recognition process. In the case of AFR, most of the undesired variability is due to lighting changes. This is especially problematic for the feature extraction step because it has been shown both empirically [1] and formally [11] that no function of an image can be illumination invariant. The feature extraction can be thought of as a function mapping images into vectors. Traditional approaches to lighting normalization include common image processing techniques such as histogram equalization, gain/offset correction and non-linear transforms of the image intensity (see [21][52] for a description of such algorithms). However, recent works show that better results, at least in the case of AFR, are obtained through biology inspired techniques, in particular the so-called *retinex* theory (see Section 13.4) [35].

The following step is the feature extraction, i.e. the conversion of the input image into a format suitable for the classification step. The feature extraction approaches can be grouped into two classes: *holistic* and *local*. In the first case, the features are extracted from the whole face image and the result is a single vector per image. In the second case, the features are extracted from different parts of the same image and the result is a sequence of vectors. The most common feature extraction techniques in holistic approaches are principal component analysis and linear discriminant analysis (see Chapter 11). Local approaches focus on specific parts of the image that are more likely to help the recognition (e.g. the eyes) and extract the features from there. In some



**Fig. 13.1.** AFR systems architecture. The figure shows the general scheme of an AFR system.

other cases, the same feature extraction technique is applied to image blocks (sometimes partially overlapping) positioned over a grid spanning the whole image. Some works simply apply PCA to local regions rather than to the whole image [46]; others involve Gabor filters [14], Gabor wavelets [34][50] or discrete the cosine transform (see Appendix B) [58].

The last step of the process is classification. In an AFR system, the goal of classification is to find the identity of the person portrayed in the input image and it is known a priori that the same person is portrayed in at least one of the pictures in the *predefined* set of images available to the classifier (see Figure 13.1). Such a task is also known as *closed set identification* in opposition to the *open set identification* where a picture of the person portrayed in the input image is not necessarily available in the predefined set of images. This Chapter focuses on the first case (closed set identification) and the classifier gives as output the identity  $I^*$  such that:

$$I^* = \arg \max_I \Lambda(X, X_I) \quad (13.1)$$

where  $X$  and  $X_I$  are the representations of the input image and of an image of identity  $I$  in the set of images, and  $\Lambda(X, X_I)$  is a score accounting for the matching of  $X$  and  $X_I$  given by the classifier (see Section 13.6 for more details). The value of  $\Lambda(X, X_I)$  is obtained using different algorithms. In some cases is a simple Euclidean or Mahalanobis distance between vectors extracted from the images. Other approaches use elastic graph matching [34], generative models [10], and discriminant models [27] such as neural networks (see Chapter 8) and support vector machines (see Chapter 9).

The next sections described in more detail each step of the process.

### 13.3 Face Detection and Localization

This section describes the face detection and localization problem. This is the first step in any technology aiming at the analysis of face images (face recognition, facial expression recognition, etc.) or using the presence of faces in an image to perform other tasks, e.g. the detection of user presence in smart interfaces. Although similar and overlapping, *detection* and *localization* are not exactly the same problem. The detection is the task of finding all faces (if any) in an image and it is not known a priori whether the image contains faces or not (and if yes how many). The localization is the task of identifying the exact position of a single face known in advance to be in the image. The main difference between detection and localization is then in the available a priori knowledge about the presence and number of faces actually appearing in each image. The localization is especially important in face recognition because many systems require the user to stand in front of a camera or are applied to passport-like pictures. The quality of the localization affects the recognition performance [41] and both detection and localization are the subject of at least two major surveys that are the basis of this section [23][71].

Following [71], the main problems in detecting and localizing faces are:

- *pose*: pan and tilt with respect to the camera
- *structural components variability*: glasses, scarfs, beards, etc.
- *expression*: smile, amazement, etc.
- *occlusion*: presence of objects between the camera and the face to be detected
- *imaging conditions*: illumination, source of lighting, camera settings, etc.

All of the approaches proposed in the literature try to deal with the above problems and their effectiveness depends in large measure on how controlled are the above factors. In the case of face recognition, pose, expression, occlusion and imaging conditions are relatively constrained. As mentioned above, the users are often required to stand in front of the camera and the most natural posture does not involve large variations in the position of the head. Moreover, the expression tends to be neutral, no occlusion is allowed, and the imaging conditions can be controlled. On the other hand, no constraint can be imposed on structural elements. This is especially difficult when the pictures of the same person are taken at large intervals of time.

The main approaches to detection and localization problems can be split into two major classes: the first includes the approaches using a priori knowledge about the so-called *facial features* (i.e. eyes, nose, lips, etc.). The techniques belonging to this class try to detect facial features in an image and then use their mutual position to infer the presence of a face. The second class includes the approaches that do not use a priori knowledge about facial features but rather try to classify each region of an image as either belonging to a face or not.

The methods using a priori knowledge are often called *knowledge-based* and typically use a top-down approach, i.e. they first analyze the image at large scale to find the regions most likely to contain a face, then they perform finer analysis on the candidate regions to detect details such as eyes, eyebrows, etc. (see e.g. [31][70]). The main problem with such approaches is that the a priori knowledge is often used under the form of rules, e.g. there must be a certain distance between the eyes, that lead to false alarms, i.e. to the detection of faces where there are other objects, when they are too flexible, but result into false negatives, i.e. they miss actually appearing faces, when they are too rigid. Moreover, the rules are often not capable of dealing with the large variability of conditions that can be found in an image.

Some knowledge-based approaches try to overcome the above problems by using the *template matching*, i.e. structures where the face elements can be moved to fit the data, e.g. the nose must lie on the direction perpendicular to the line connecting the eyes, but such a condition can be relaxed to a certain extent by deforming the template (see e.g. [13][16]). The main problem of such approaches is that they cannot deal effectively with variations in scale, pose and shape [71].

Techniques classifying image regions as either belonging or not to a face are typically based on a bottom-up approach, i.e. they infer the presence of a face starting from low-level features not influenced by pose and scale variations as well as lighting variations and other sources of undesired variability (see above). Some techniques try identify the regions most likely to corresponds to facial features by using edge detection techniques (i.e. sudden change regions) [12], connected components in gray-level images [17], local feature extractors and random graph matching [36], etc. Many other approaches are based on the identification of the textures most likely to corresponds to human faces or to recognize the skin color (see [23][71] for a wide bibliography).

This class of approaches includes also the so-called *appearance-based* methods, i.e. techniques that learn from large set of images to distinguish between face and nonface regions. This is the most recent recent trend and the results are good compared to the previous approaches. The only problem is that machine learning approaches require large amounts of labeled data (often each image pixel must be labeled separately) and this can be an obstacle. Most of the algorithms presented in the previous chapters have been used for the face detection and localization problems: principal component analysis [30][64], neural networks [8][62], support vector machines [19][45] and hidden Markov models [56].

The performance of a detection and localization systems is measured through the percentage of faces correctly identified out of a test set of images or videos. However, such information alone is not enough because it takes into account only *false negatives*, i.e. nondetection of faces actually appearing in the data, while detection systems perform also another kind of error, i.e. the *false positive*, the detection of a nonexistent face. Such a figure must then be



included in the evaluation. On the other hand, since detection and localization are typically the first step of a longer process, recent works suggest to evaluate the localization through the impact on the end-application [54]. In other words, the best system is not the one that best locates faces, but the one that results into the best recognition or identification performance.

### 13.3.1 Face Segmentation and Normalization with *TorchVision*

The *TorchVision* package contains a face segmentation tool based on the eyes position. When the eyes position is known, the segmentation plays the role of the face localization. The typical command line is as follows (see the package website for more details and for sample data):

```
faceExtract inputImg.pgm inputImg.pos -facemodel 3 -oneface
          -postype 1 -norm -savebin outputImg.bindata
```

where `inputImg.pgm` is the input image in pgm format (see Chapter 3), `inputImg.pos` is the file containing the position of the eyes, and the options have the following effects:

- `facemodel` specifies the image dimensions (in the example, the value 3 leads to 64×80 pixels output images).
- `oneface` specifies that the input image contains only one face.
- `postype` specifies the format of the eyes position file.
- `norm` specifies that the pixel values of the output image will be normalized between 0 and 1.
- `savebin` specifies the name and the format of the output image.

Figure 13.2 shows the results of the function on a sample image. The circles in the input image are the position of the eyes as given in the `inputImg` file, and the smaller images on the right side show the output image both before and after the normalization of the pixel values (see documentation on the package site for more information about available options and their effect).

## 13.4 Lighting Normalization

The goal of lighting normalization is to eliminate the variability due to illumination differences between images. While the other steps of the AFR process are performed with many different approaches, the normalization is performed with relatively few standard techniques. Traditional methods include histogram equalization, gain/offset correction, nonlinear transforms (e.g. logarithmic) of the image intensity and homomorphic filtering (all the algorithms are described in [52]). However, bio-inspired techniques based on the so-called *retinex theory* [35] have been shown in recent years to perform better than the above algorithms [60][9]. For this reason, this section focuses



**Fig. 13.2.** Face localization. The picture shows how the face is first localized and then normalized out of the original image (courtesy of Sébastien Marcel).

on two retinex-based algorithms known as *center/surround retinex* [25][26], and *Gross and Brajovic's* (GB) [18].

Following the retinex theory, an image  $I(x, y)$  can be reconstructed as follows:

$$I(x, y) = R(x, y)L(x, y) \quad (13.2)$$

where  $R(x, y)$  is the reflectance, i.e. the fraction of incident light energy reflected at point  $(x, y)$ , and  $L(x, y)$  is the lighting map underlying the image, i.e. the function giving the incident light energy at point  $(x, y)$ . The normalization can be thought of as a process which transforms the lighting map of all images into the same target function  $L^*(x, y)$ , but such an operation can be performed only after reconstructing  $R(x, y)$ . For this reason, retinex theory based algorithms focus on the estimation of the reflectance map. The next two sections show how this is done in Center/Surround Retinex and Gross and Brajovic's algorithms.

### 13.4.1 Center/Surround Retinex

The model proposed in [25][26] estimates the reflectance in a pixel  $(x, y)$  as a weighted average of the surrounding pixels. The first version of the algorithm [26] performs the average at a single scale and estimates the reflectance as follows:

$$R(x_0, y_0) = \log I(x_0, y_0) - \log[I(x, y) * G_\sigma(x, y)] \quad (13.3)$$

where  $G_\sigma(x, y)$  is a Gaussian filter (GF) of variance  $\sigma^2$ . The GF has the following form:

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right) \quad (13.4)$$

and its application results in a blurred version of the original image. The rationale behind such an approach is that a weighted sum of the intensities surrounding a pixel provides a better estimation of the same pixel. The use of the logarithm in Equation (13.3) corresponds to the logarithmic relationship between intensity and human eye perception of intensity (see Chapter 3).

The same algorithm has been proposed in a multiscale version in [25]:

$$R(x, y) = \sum_{\sigma=1}^S (\log I(x_0, y_0) - \log[I(x, y) * G_{\sigma}(x, y)]) \quad (13.5)$$

where the use of several values of  $\sigma$  enables one to deal with both uniform, changes can be observed only at large scales with high  $\sigma$  values, and variable, changes can be observed at small scales with low  $\sigma$  values, illumination maps depending on the region.

### 13.4.2 Gross and Brajovic's Algorithm

The GB algorithm estimates the luminance  $L(x, y)$  by minimizing the following expression:

$$E(L) = \int \Omega_x \int \Omega_y \rho(x, y) [L(x, y) - I(x, y)]^2 dx dy + \lambda \int \Omega_x \int \Omega_y (L_x^2 + L_y^2) dx dy \quad (13.6)$$

where  $\Omega_x$  and  $\Omega_y$  are the  $x$  and  $y$  domains,  $\rho(x, y)$  is the diffusion coefficient,<sup>2</sup>  $\lambda$  is a parameter weighting the importance of the second integral, and  $L_x$  and  $L_y$  are the derivatives of  $L$  with respect to  $x$  and  $y$ . The first term of Equation 13.6 accounts for the similarity between  $I(x, y)$  and  $L(x, y)$ , while the second one is a smoothing term.

### 13.4.3 Normalization with *TorchVision*

The *TorchVision* function for the normalization performs an histogram equalization and a Gaussian smoothing (see [40] for details). The command line is as follows:

```
binfacenormalize inputImg.bindata 64 80 -unnorm -mirroradd -norm
-o output.bindata
```

where `inputImg.bindata` is the input image (in `bindata` format), 64 is the number of image columns, and 80 is the number of image rows. The option effects are as follows:

- `unnorm` specifies that the pixel values of the input image are normalized between 0 and 1.

<sup>2</sup> The *diffusion coefficient* is a factor of proportionality representing the amount of energy diffusing across a unit area through a unit energy gradient in unit time.

- `mirroradd` specifies that the mirror image of the input face is added to the output image (this helps the recognition and verification performance).
- `norm` specifies that the pixel values of the output image will be normalized between 0 and 1.
- `o` stands for output and specifies the name of the output image.

The results of the function are in Figure 13.2 (lower image on the right side).

## 13.5 Feature Extraction

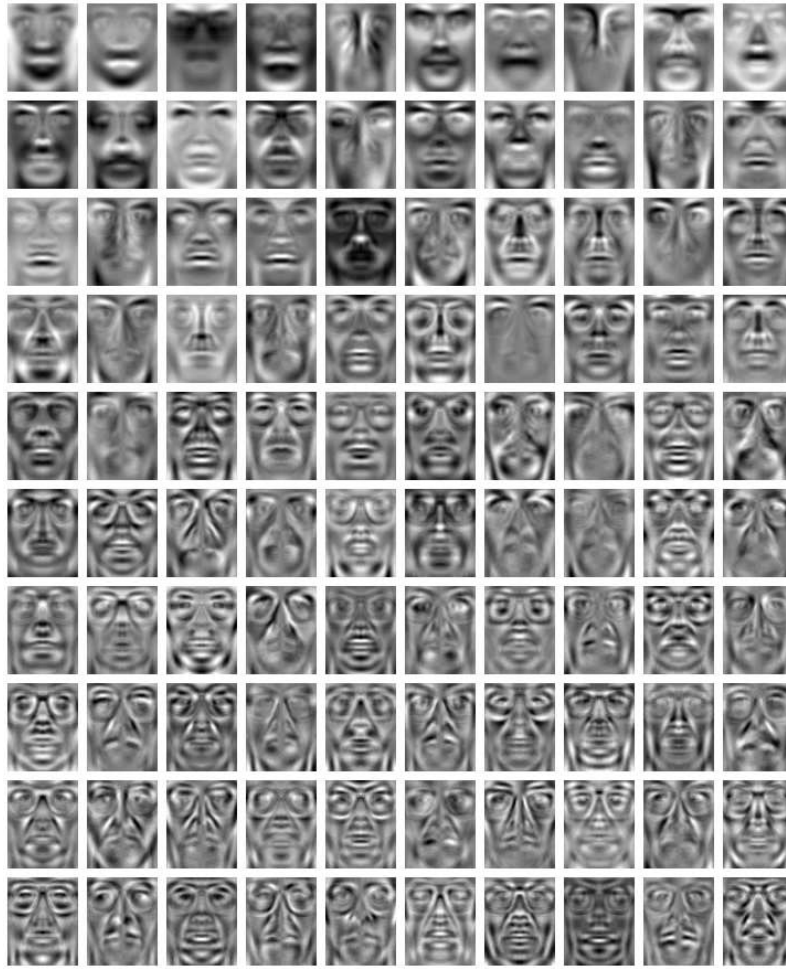
This section presents some feature extraction methods frequently applied in AFR. Special attention is paid to the application of principal component analysis (see Chapter 11) and the extraction of the so-called eigenfaces. Such an approach is one of the earliest of the literature, but it is still today used as a baseline for comparison with other techniques [72]. Feature extraction techniques for face recognition can be broadly grouped into two classes: *holistic approaches* and *local approaches* (see Section 13.2 for more details). In the first case, the face image is converted into a single vector resulting from the application of an algorithm to the whole image face. In the second case, the face image is converted into a set of vectors extracted from selected regions of the image. The next two subsections show the two above approaches in more detail.

### 13.5.1 Holistic Approaches

This section presents the main holistic feature extraction approaches with a special attention to the PCA. The reason is not only that the PCA is often used as a baseline for comparison with other approaches, but also that the eigenvectors can be visualized. This provides a rather clear *visual* example of how the PCA works, i.e. of how the most important information is captured by projecting the images onto the eigenvectors.

Proposed for AFR in [30][61], the PCA has been applied for the first time in [64] resulting into the first successful AFR system [44][48][65][72]. The rationale behind the application of the PCA is that natural images tend to be redundant, especially when they contain the same object and are the output of a normalization process [47][55], then the PCA is a suitable representation because it decorrelates the data and enables one to capture most of the information contained in the faces using few features.

In holistic approaches, each image is considered as a point in the data space. In other words, the images are considered as vectors where each component corresponds to a pixel. Average face images contain several thousands of pixels (Section 13.8 shows examples where the face images contain 5120 pixels), then they cannot be fed directly to a classifier because of the *curse of dimensionality* (see Section 11). The application of the PCA can signifi-



**Fig. 13.3.** Eigenfaces. The figure shows the 100 eigenfaces corresponding to the 100 eigenvectors with higher eigenvalues extracted from a face image database.

cantly reduce the number of features necessary to represent the same data. Figure 13.3 shows the first hundred eigenvectors extracted from the training set of the XM2VTS database (see Section 13.8), one of the benchmarks most commonly used in the literature. The eigenvectors are the basis of the space of the face images; then they are face images as well. For this reason it is possible to visualize them and to see *ghostlike* faces often called *eigenfaces* (see Figure 13.3) [64]. The clear areas correspond to higher components, i.e. to the face regions that are more weighted when a face image is projected onto the eigenvectors. The first eigenface seems to account, not surprisingly, for eyes, nose and upper lips; the second one seems to account especially for

the lower part of the mouth; the third one corresponds to the eyebrows area, and so on. This provides an intuitive explanation of where most of the face variance is concentrated.

Figure 13.4 shows the percentage of data variance retained as a function of the number of eigenvectors. The first 20 eigenvectors correspond to more than 50% of the data variance, but to reach 90% it is necessary to include around 250 eigenfaces (this point is not plotted in the figure). This is evident in Figure 13.5 where several faces are reconstructed using 10%, 20%, . . . , 100% of the data variance. Each reconstructed image is the linear combination of the number of eigenvectors corresponding to a given value of variance. The coefficients of the linear combination are the projections of the original images onto the corresponding eigenvectors. The difference between the different images becomes evident only at 50% of the variance and this is confirmed by the recognition results presented in Section 13.8. In fact, the percentage of faces correctly recognized increases quickly up to a number of eigenvectors corresponding to 50-60% of the variance, and then increases slowly as more information is added. This means that the first eigenvectors contain most of the information while the others give less and less significant contributions (see Section 13.8 and Figure 13.11 for more details).

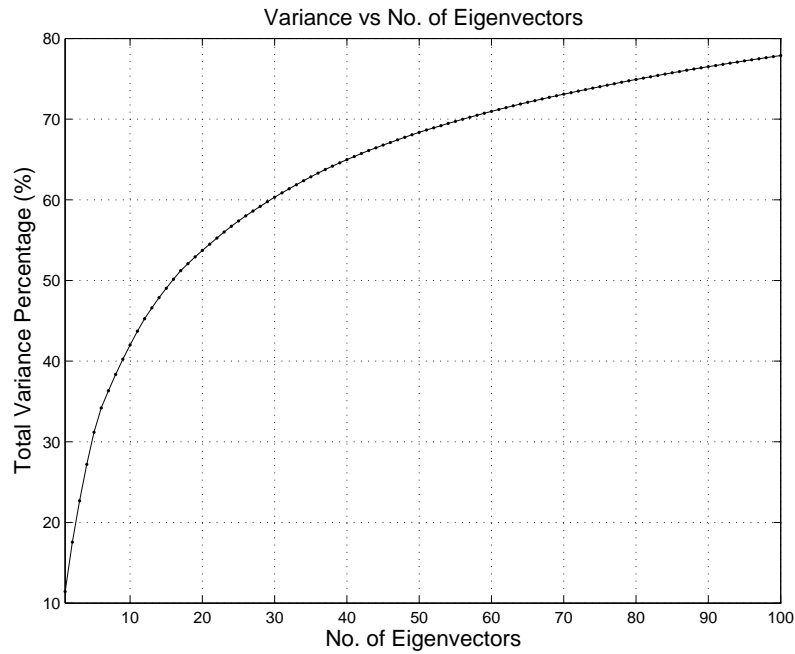
The results show that, in such a representation, less than 30 features (the projection onto the first eigenfaces) are sufficient to achieve satisfactory results [72]. The original dimension of the images is then reduced by around 170 times and the application of the classifiers presented in the previous chapters is possible.

The earliest approaches performed the recognition by simply finding the nearest neighbor in the set of images at disposition (see Figure 13.1) [64]. Such an approach has then been refined by applying a Bayesian approach [43], increasing the amount of data at disposition [50] and by trying to identify subspaces more informative than others [62].

The PCA is just one way to convert high dimension vectors into lower dimension data preserving most of the information. The other approaches are presented in detail in Chapter 11 and have been often applied in face recognition, in particular *linear discriminant analysis* with different variants [5][15][39] [42][63][73] and independent component analysis [3][4]. Other feature extraction approaches are based on genetic algorithms [37] and kernel methods [2][38][71][69][75][74].

### 13.5.2 Local Approaches

Local approaches do not convert face images into a single vector, but rather into a sequence of vectors extracted from regions supposed to be more informative. In some cases, the feature extraction techniques applied to single regions are the same as those applied to the images as a whole in holistic approaches, e.g. the local PCA in [46], but in most cases, local approaches use different kinds of feature extraction techniques. The two-dimensional Gabor



**Fig. 13.4.** Variance. The plot shows the value of the data variance captured as a function of the number of retained eigenvectors.

wavelets [14] have been successfully applied in [34][50]. The authors of such works overimpose a grid over a face image and identify the pixels  $(i_0, j_0)$  corresponding to the grid nodes. For each node  $(i_0, j_0)$  they extract a feature vector where the components are the output of Gabor filters with different directions and scales.

Other approaches split the images into nonoverlapping blocks and apply a discrete cosine transform (see Appendix B) to each one of them [20][58][59] or to the blocks containing more informative regions like the eye area [22]. The feature vector sequences resulting from the different local areas are typically modeled using hidden Markov models (see Chapter 10) [10].

### 13.5.3 Feature Extraction with *TorchVision*

The *TorchVision* package implements three feature extraction methods: PCA and LDA (suitable for holistic approaches), and DCT (suitable for local approaches). The features are extracted from the face images after the application of localization and normalization functions (`faceExtract` and `binfacenormalize`, respectively).

The PCA extraction is performed with the following command example:

```
trainPCA list.dat n -verbose -save model.pca
```



**Fig. 13.5.** Reconstruction. Each row shows how a face image is reconstructed using an increasing number of eigenvectors. The first image (starting from the left) uses 10 eigenvectors; the second uses 20 eigenvectors; and so on. The last image of each row is the original picture.

where `list.dat` is the list containing the names of the files to be processed and `n` is the number of pixels in the images of `list.dat`. The effect of the options is as follows:

- `verbose` specifies that the program provides output about the steps being performed at each moment.
- `save` specifies the file where the principal components must be stored (in the case of the example the name of the output file is `model.pca`).



The eigenvalues can be converted into images for visualization purposes with the following function:

```
pca2pgm model.pca nCol nRow -eigenface n -verbose
```

where `model.pca` is the file containing the principal components (output of the function `trainPca`), and `nCol` and `nRow` are the number of columns and rows respectively in the images from where the PCs have been extracted. The effect of the options is as follows:

- `eigenface` specifies the number of eigenvectors to be converted into images (`n` in the example).
- `verbose` specifies that the program provides output about the steps being performed at each moment.

The results are shown in Figure 13.3. The images can be projected onto the the eigenfaces to produce compact feature vectors (see Section 13.5). This can be done in two possible ways: the first is by projecting the images onto a predefined number of eigenfaces, the second is by projecting the images onto a number of eigenfaces accounting for a predefined fraction of data variance (see Section 13.5).

The function for the projection onto a predefined number of eigenfaces is as follows:

```
bindata2pca inputImg.bindata model.pca n -noutput nEig -o
output.bindata
```

where `inputImg.bindata` is the image to be projected onto the eigenvectors (in `bindata` format), `model.pca` is the file containing the principal components (obtained with function `trainPca`), and `n` is the number of pixels in the input images. The effect of the options is as follows:

- `noutput` specifies the number (`nEig` in the example) of eigenvectors to be used.
- `o` stands for output and specifies the name of the output data.

The function for the projection onto a number of eigenfaces accounting for a predefined fraction of the data variance is as follows:

```
bindata2pca inputImg.bindata model.pca n -variance fVar -o
output.bindata
```

and the meaning of the symbols is the same as in the case of the projection onto a predefined number of eigenvectors (see above). The effect of the options is as follows:

- `variance` specifies the fraction (`fVar` in the example) of variance to be used (it must be a number between 0 and 1).
- `o` stands for output and specifies the name of the output data.

The results of the projection are shown in Figure 13.5. The picture shows that the higher the number of eigenvectors, the better are the approximation of the original image.

The images represented in the PCA space can be used to extract linear discriminant analysis features using the following command:

```
trainLDA fileList.dat n -n_classes 200 -save model.lda
```

where `fileList.dat` is the list containing the names of the reconstructed images to be used, and `n` is the number of eigenvectors onto which the images have been projected. The effect of the options is as follows:

- `n_classes` specifies the number of classes in the data (in this case the number of identities).
- `save` specifies the name of the file where the LDA subspace axes have been stored (`model.lda` in the example).

Once the LDA has been performed, the reconstructed images can be projected onto the LDA subspace axes to obtain new feature vectors. The command is as follows:

```
bindata2lda inputImg.bindata model.lda n -n_output m -o
outputImg.bindata
```

where `inputImg.bindata` is the input image in `bindata` format (see above), `model.lda` is the file containing the LDA axes (see function `trainLDA`) and `n` is the number of eigenvectors onto which the reconstructed image `inputImg` has been projected. The effect of the options is as follows:

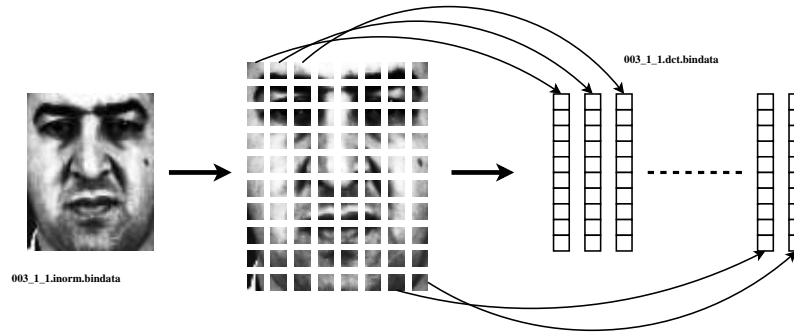
- `n_output` specifies the number of LDA axes onto which the input image must be projected, i.e. the dimension of the output vector.
- `o` specifies the name of the output file.

PCA and LDA are the two holistic feature extraction approaches implemented in *TorchVision*. The package implements also the Discrete Cosine Transform through the following command:

```
bindata2dctbindata inputImg.bindata m n outputImg.dct.bindata
-unnorm -block b -overlap o -dctdim d
```

where `inputImg.bindata` is an input image in `bindata` format (see above), `m` and `n` are the number of columns and rows in the image, respectively, and `outputImg.dct.bindata` is the name of the file containing the feature vector. The effect of the options is as follows:

- `unnorm` specifies that the input image is normalized (pixel values between 0 and 1) and must be converted into a gray-level image (pixel values between 0 and 255).
- `block` specifies the size of the square blocks from which the DCT is extracted (`b` in the example).



**Fig. 13.6.** DCT feature extraction. The image shows how the DCT features are extracted from different blocks of the image (courtesy of Sébastien Marcel).

- `overlap` specifies the overlap between neighboring blocks (`o` in the example).
- `dctdim` specifies the number of DCT coefficients to be retained, i.e. the dimension of the feature vectors (`d` in the example).

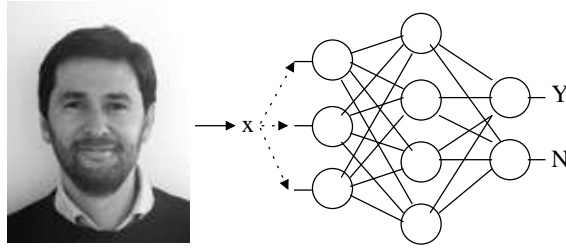
The feature extraction approach implemented by `bindata2dctbindata` is shown in Figure 13.6.

## 13.6 Classification

The classification step depends on the feature extraction approach: holistic methods use classifiers such as neural networks or SVM that can be thought of as mappings between the space of the faces and the space of the identities, while local approaches use hidden Markov models which can provide the likelihood for sequences of vectors given a face model. However, the goal of the classification step is the same in both cases: given a set of images (see Figure 13.1)  $F$  and an input image (not belonging to  $F$ ), the classifier must find the image in  $F$  which portrays the same person as the input image. In general,  $F$  contains more than one image per person in order to account for the variability due to pose, ageing, hair style, etc.

This chapter focuses on *close set* recognition, i.e. the person in the input image is supposed to be represented in  $F$ . The problem is then similar to speech and handwriting recognition (see Chapter 12) where the recognizer can give as output only words appearing in the dictionary. The two main problems for a classifier in AFR are that few examples (typically less than 10) per identity are available and that the number of output classes (i.e. of identities) is high (several hundreds).

The earliest approaches (e.g. [64]) were based on a nearest neighbor approach: the images of  $F$  and the input images are converted into vectors. Then, given an input vector  $\mathbf{x}$ , the system assigns the identity of the image



**Fig. 13.7.** Neural networks classification. The feature extraction is fed to a neural network which has an output for each identity. The network is trained to have positive output for the correct identity and negative output for the others.

corresponding to the vector  $\mathbf{f}^*$  such that:

$$\mathbf{f}^* = \arg \min_{\mathbf{f} \in F} (\mathbf{f} - \mathbf{x})^2. \quad (13.7)$$

However, once the size of the available data sets  $F$  has started to grow, it is possible to apply classifiers such as neural networks and support vector machines which need to be trained on a sufficient amount of data in order to perform correctly. Neural networks are trained over the images of  $F$  and associate to an input image one of the identities represented in  $F$ . In other words, the NN correspond to a mapping  $f$  capable of associating identities to face images:

$$f : \mathcal{X} \rightarrow \mathcal{I} \quad (13.8)$$

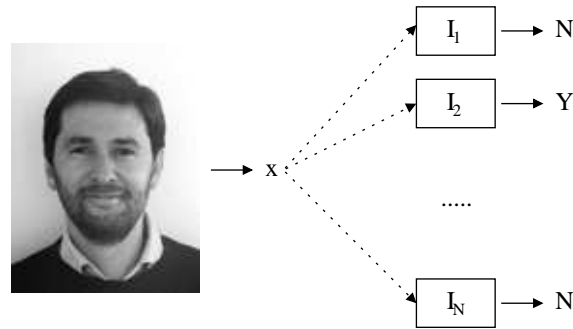
where  $\mathcal{X}$  is the data space and  $\mathcal{I}$  is the list of the identities represented in  $F$  (see Figure 13.7).

Since the SVMs are binary classifiers, it is necessary to train a different SVM for each identity in  $\mathcal{I}$ . The SVMs are supposed to provide positive scores to images portraying the same person they correspond to and negative scores to the others. In case of multiple positive answers, the tie can be broken by selecting the identity of the SVM giving the highest score:

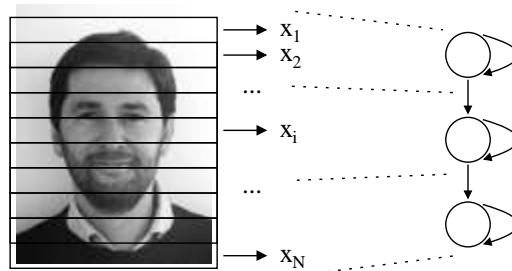
$$I^* = \arg \max_{I \in \mathcal{I}} \alpha_I(\mathbf{x}) \quad (13.9)$$

where  $I$  is an identity and  $\alpha_I(\mathbf{x})$  is the score that the SVM corresponding to  $I$  assigns to  $\mathbf{x}$  (see Figure 13.8).

While Chapter 12 has shown that the same machine learning approach can be applied to different data, this section shows that different machine learning approaches can be applied to the same data. Given a representation of a face image (PCA, ICA, etc., see Section 13.5), different algorithms can be applied for the classification. The same applies to the use of HMMs in local approaches. Chapter 12 shows how HMMs model sequences of feature vectors extracted from spoken and handwritten words and the HMMs can be used in the same way to model vectors sequences extracted from face images (see Figure 13.9).



**Fig. 13.8.** SVM classification. The feature vector is fed to  $N$  SVMs corresponding to the  $N$  possible identities. Each SVM is trained to give positive answer only for images portraying persons of the same identity they correspond to.



**Fig. 13.9.** HMM classification. An HMM is trained for each identity so that it is possible to know the likelihood of the vector sequence given each identity model.

### 13.7 Performance Assessment

The performance assessment problem has two basic requirements: the first is the definition of a performance metric capable of measuring correctly the effectiveness of the system; the second is the definition of standard benchmarks, i.e. common data and experimental protocols used by the whole research community. The advantage of the second aspect is that results obtained by different groups can be compared rigorously. While the first condition is typically met in any domain, good performance measures are available for every application, the realization of the second condition is the exception rather than the rule. AFR is one of the exceptions and at least two major benchmarks (FERET and FRVT) are available to the researchers. This is an important advantage because the different techniques proposed in the literature can be compared in the same conditions and rigorous answers about the effectiveness of one or the other can be obtained.

The next sections present the different benchmarks and show, whenever possible, the results achieved using different approaches.



**Fig. 13.10.** FERET database. The figure shows images from a set (faces 1 to 3) and from its duplicate (faces 4 and 5). The duplicate faces have been photographed two years after the first three

### 13.7.1 The FERET Database

The *FERET* database has been collected at the *National Institute of Standard and Technology* (NIST) in the United States and aims at benchmarking two major applications: AFR and *automatic face verification* (AFV), i.e. the process of accepting or rejecting the identity claim made by a person (typically called *client*). This section focuses on AFR results [50][51], but results of the AFV assessment are available in [53].

The database contains 14,126 images split into 1,564 *sets* [50]. Each set contains 5-11 images of the same person taken in different conditions, e.g. with and without glasses or with different facial expressions. The total number of identities is 1,199 and 365 sets are made of *duplicates*, i.e. of images of a person represented in another set, but at a different moment. In some cases there are two years of difference between the pictures of one set and the pictures of the duplicate. The FERET database aims at reproducing the so-called *law-enforcement* scenario where one person is asked to identify a suspect in a collection of pictures showing frontal faces of previously arrested people. From an AFR point of view the above scenario has two main problems: high number of classes (1,199) and few training samples for each class (5-11).

The latest FERET tests were performed as follows: each participating team is provided with two sets of images: the *target set* (3,323 images) and the *query set* (3,816 images). Both query and target sets were not available in the training phase, then at the moment of the test both sets are not known to the systems. Given an image in the query set, the recognition systems find the best matching image in the target set. If the query image and best matching target image show the same person, then the recognition process is correct. The percentage of query images for which the recognition process is correct is the performance metric of the test. Several teams have participated in the test: Massachusetts Institute of Technology (MIT) [43][64] Michigan State University (MSU) [63][73], Rutgers University (RU) [67], University of South California (USC) [68] and the University of Maryland (UM) [15]. Some of the teams (MIT and UM) participated with more than one system. The complete results are available in [50], tests were performed in different con-

ditions and using different protocols to highlight different properties of the systems. Overall, the best system is the one described in [43] and based on holistic PCA representation and Bayesian approaches for the classification.

The FERET database is still available at NIST and, although no more official evaluations have been carried out, is still used today as a benchmark in many works of the literature.

### 13.7.2 The FRVT database

The FRVT (*face recognition vendor test*) database and related official tests can be considered as the continuation of the FERET evaluation campaigns.<sup>3</sup> Five companies participated in the evaluations (see [6][72] for their names), hence the name *vendor test*. The main goal of FRVT is to investigate the problems left open by FERET, i.e. the effect of on the performance of the following effects: different facial expressions, use of lossy image compression algorithms, distance of the face from the camera, lighting changes, media used for image storing (CCD rather than film), head pose, image resolution and delay between different images [6][7].

Each of the above effects has been investigated by creating an appropriate dataset (often including FERET data). The results are reported in [7] (in extensive form) and [72] (in concise form). The finding of the evaluation can be summarized as follows:

- *Effect of compression rate.* No statistically significant changes are observed for compression rates up to 30:1. The recognition rate decreases from 63% to 56% when compressing the images 40 times using the JPEG algorithms (see Chapter 3). This is important for the applications running on portable devices such as cellular phones (e.g. identification of the owner), or through the web (e.g. remote recognition for accessing web based services).
- *Effect of the media.* The results obtained using digital cameras and 35mm films are similar. This is important in applications like indexing and content analysis of journals and other printed materials.
- *Effect of the expression.* The expression affects slightly (less than 3%) the recognition rate. This is important when the subject cannot be asked to have a neutral expression (e.g. in personal albums).
- *Effect of lighting.* The effect of the lighting is significant, more than 30% of change, especially when moving from indoor to outdoor where the illumination cannot be controlled. This seems to suggest that for the moment recognition applications must be limited to controlled environments.
- *Effect of pose.* The pose is the angle by which the head is rotated with respect to the camera. The results show slight changes (less than 5%) when the pose is in the interval  $[-25^\circ, 25^\circ]$ , but major recognition rate decreases (more than 50%) when the pose is higher than  $40^\circ$ . In other

---

<sup>3</sup> At the time this book is being written, the informations about past and future FRVT evaluations are available at the following site: [www.frvt.org](http://www.frvt.org).

words, the frontal pose is not a strict requirement, but no major deviations with respect to such a conditions are allowed.

- *Effect of resolution.* The resolution is measured through the number of pixels separating the two eyes, this roughly accounts for the number of pixels on the face of the subject. Moderate changes (less than 5%) are observed when passing from 60 to 15 pixels, with the exception of two participating systems.
- *Effect of time delay.* The results show no major changes when recognizing face images separated by up to two years. This is important because the set of available images must not be updated too frequently.

While they provide excellent indications about the limits of AFR technologies, the FRVT results do not give any hint about the algorithms used by the different systems. The reason is that the vendors participating in the evaluations keep the details of their products confidential.

## 13.8 Experiments

This section proposes some experiments that can be easily implemented using *TorchVision* and *svmLight*, the support vector machine package presented in Chapter 9. The goal is not to achieve state-of-the-art results because more recent approaches achieve better performances, but rather to suggest some laboratory exercises based on material accessible on the web. The first three steps of the processing (localization, normalization and feature extraction) are performed using the *TorchVision* functions described in the previous part of the chapter and this section focuses solely on the classification step. All the experiments we perform are based on PCA features, but the reader can repeat the experiments using other features to compare the results. Two classification approaches are used: the first is the simple Euclidean distance between the input faces and the faces available in the training set, the second is based on support vector machines. The experiments are performed over the *XM2VTS* database,<sup>4</sup> but it is possible to use other data.

The next sections describe the data, the results obtained using the Euclidean distance and the results obtained using the SVMs.

### 13.8.1 Data and Experimental Protocol

The experiments described in this section are based on the *XM2VTS* database, a multimodal collection of face images and videos accompanied by speech samples of each portrayed individual. The experiments of this section use only the

<sup>4</sup> At the time this book is being written the data is available at cost price at the following website: <http://www.ee.surrey.ac.uk/Research/VSSP/xm2vtsdb>. Feature vectors extracted from the database are available on <http://TorchVision.idiap.ch/documentation.php>, following the link *Examples or Labs*.



face images and all other data are not considered. The face image data set contains 2333 samples showing 295 individuals. All individuals participated in four capture sessions where they have been photographed two times. As a result, there are eight pictures per person, with the exception of few individuals who could not participate in all sessions. The images of the first three sessions are used as a training set (for a total of 1747 faces), while the others are used as test set (for a total of 586 faces). Some samples of the database are shown in Figure 13.5 (rightmost column) as a result of the projection onto all eigenfaces extracted from the training set. Each person appearing in the test set is represented also in the training set and the total number of identities is 295.

### 13.8.2 Euclidean Distance-Based Classifier

The classification based on Euclidean distance is probably the easiest possible approach to the problem of face recognition. If  $\mathcal{X}$  is the set of the feature vectors extracted from the training set (see previous section) and  $\mathbf{y}$  is the feature vector extracted from the face to be recognized, then the classification step simply finds the vector  $\mathbf{x}^* \in \mathcal{X}$  such that:

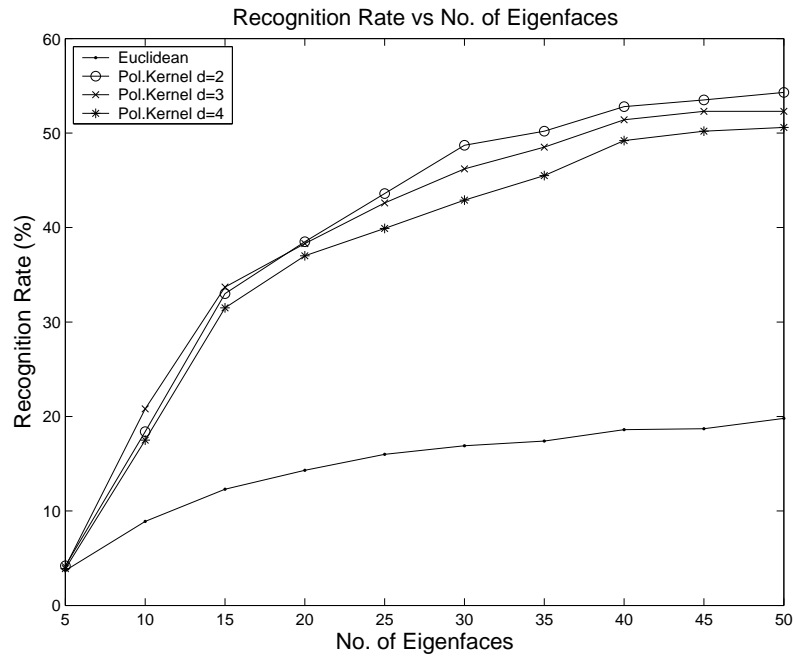
$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{X}} (\mathbf{x} - \mathbf{y})^2. \quad (13.10)$$

If  $I(\mathbf{x})$  is the identity of the face from where  $\mathbf{x}$  has been extracted, then  $I(\mathbf{x}^*)$  is assigned to  $\mathbf{y}$ .

In the case of our experiments, the vectors are the projections of the images onto the first  $D$  eigenfaces extracted from the training set. If the vectors  $\mathbf{y}$  and  $\mathbf{x}^*$  are extracted from images of the same person, then  $\mathbf{y}$  is correctly recognized. The performance measure is simply the percentage of images in the test set that have been correctly recognized.

The value of  $D$  is typically set by preserving a certain amount of variance (typically 90%), but for didactical purposes our experiments are performed varying  $D$  from 5 to 50 step 5. The goal is to show how the recognition performance changes as a function of the amount of variance retained and that relatively good recognition performances can be achieved even with few eigenvectors. In realistic settings, the value of  $D$  must be set through cross-validation, i.e. by selecting the value that give satisfactory results on a set of data independent from the training and from the test set.

The results of the experiments are reported in Figure 13.11 where the plot shows the recognition rate as a function of  $D$ . After growing relatively fast at the first steps, the recognition rate is multiplied by more than four when passing from  $D = 5$  to  $D = 25$ , the curve increases more slowly. The reason is that the amount of useful information brought by the eigenfaces is lower and lower when increasing  $D$  (see Figure 13.4). The plot stops at  $D = 50$  and this accounts for around 70.0% of the data variance, but the performance values can be measured also for higher  $D$ . When around 90.0% of the variance is



**Fig. 13.11.** Euclidean distance performance. The plot shows the recognition rate as a function of the number of eigenfaces used to represent the data.

retained ( $D = 250$ ), the recognition rate is 22.9%, just three points more than the performance at  $D = 50$ .

Such results are far from the state-of-the-art (just see SVM-based experiments for better results), but we might wonder how good (or bad) they are in absolute terms, i.e. independently of the comparison with other systems. One of the most common ways of answering such a question is to estimate the performance of a system working *randomly*, i.e. a system that gives as output an identity drawn with uniform probability distribution from the set of the possible identities. For such a system, the probability  $p_0$  of correctly identifying an image is:

$$p_0 = \frac{1}{N}, \quad (13.11)$$

where  $N$  is the total number of identities in the dataset. In our case,  $N = 295$  and  $p_0 \simeq 0.3\%$ , then the system performs around 10 times better than chance even when  $D = 5$  (the ratio rises to 76.3 when  $D = 50$ ). The ratio between the actual performance of a system and the performance of a system operating randomly is a good metric to assess the actual effectiveness of the system. In fact, if the performance of a system is comparable to the performance of a random guess, the results are due to chance rather than to the actual effect of the algorithms.

### 13.8.3 SVM-Based Classification

In the experiments reported in this section, a different SVM is trained for each of the 295 identities represented in the training set. The SVM related to identity  $i$  is trained to give positive answer when the person appearing in the probe image  $\mathbf{y}$  has identity  $i$  and negative answer otherwise. If  $\alpha_i$  is the score that the SVM related to identity  $i$  assigns to  $\mathbf{y}$ , then the classification step finds the identity  $k$  such that:

$$k = \arg \max_{i \in \{1, \dots, N\}} \alpha_i \quad (13.12)$$

and  $\mathbf{y}$  is assigned the identity  $k$ . The SVMs are trained and tested using the *SVMLight* package which implements different kernels. In the experiments presented in this chapter, we used the polynomial kernel with degree  $d = 2$ ,  $d = 3$  and  $d = 4$ . The recognition performance is measured as in the case of the Euclidean-distance based classifier and Figure 13.11 shows the recognition rate as a function of  $D$ .

The results are similar to those obtained with the distance-based classifier only for  $D = 5$ . This seems to suggest that the amount of information the first eigenfaces account for is too small for any classifier. However, the difference becomes significant at  $D = 10$  and never stops to grow as  $D$  increases. The difference between kernels of various degree is not significant and the three curves are close to each other. Like in the case of the distance based classifier, the curve is steep for low values of  $D$ , when few eigenfaces add a significant amount of variance, and then increases at a lower rate. The recognition rate for  $D = 250$  (roughly 90% of the data variance) is 61.8 percent for  $d = 2$ , 60% for  $d = 3$  and 57.4% for  $d = 4$ .



---

## References

1. Y. Adini, Y. Moses, and S. Ullman. Face recognition: the problem of compensating for changes in illumination detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):721–732, 1997.
2. F.R. Bach and M.I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.
3. M.S. Bartlett, H.M. Lades, and T. Sejnowski. Independent component representation for face recognition. In *Proceedings of SPIE Symposium of Electronic Imaging: Science and Technology*, pages 528–539, 1998.
4. M.S. Bartlett, J.R. Movellan, and T. Sejnowski. Face recognition by Independent Component Analysis. *IEEE Transactions on Neural Networks*, 13(6):1450–1464, 2002.
5. P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:711–720, 1997.
6. D.M. Blackburn, J.M. Bone, and P.J. Phillips. FRVT 2000 executive overview. Technical report, [www.frvt.org](http://www.frvt.org), 2000.
7. D.M. Blackburn, J.M. Bone, and P.J. Phillips. FRVT 2000 evaluation report. Technical report, [www.frvt.org](http://www.frvt.org), 2001.
8. G. Burel and D. Carel. Detection and localization of faces on digital images. *Pattern Recognition Letters*, 15(10):963–967, 1994.
9. F. Cardinaux. *Face Authentication based on local features and generative models*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2005.
10. F. Cardinaux, C. Sanderson, and S. Bengio. User authentication via adapted statistical models for face images. *IEEE Transactions on Signal Processing*, 54(1):361–373, 2006.
11. H. Chen, Belhumeur, and D. Jacobs. In search of illumination variants. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 254–261, 2000.
12. D. Chetverikov and A. Lerch. Multiresolution face detection. *Theoretical Foundations of Computer Vision*, 69:131–140, 1993.
13. I. Craw, H. Ellis, and J. Lishman. Automatic extraction of face features. *Pattern Recognition Letters*, 5:183–187, 1987.
14. J. Daugman. Uncertainty relation for resolution in space, spacial frequency and orientation optimized by two-dimensional visual cortical filters. *Journal of Optical Society of America*, 2(7), 1985.

15. K. Etemad and R. Chellappa. Discriminant analysis for recognition of human face images. *Journal of Optical Society of America*, A14:1724–1733, 1997.
16. V. Govindaraju. Locating human faces on photographs. *International Journal of Computer Vision*, 19(2):129–146, 1996.
17. H.P. Graf, E. Cosatto, D. Gibbon, M. Kocheisen, and E. Petajan. Multimodal system for locating heads and faces. In *Proceedings of Second International Conference on Face and Gesture Recognition*, pages 88–93, 1996.
18. R. Gross and V. Brajovic. An image preprocessing algorithm for illumination invariant face recognition. In *Proceedings of International Conference on Audio and Video Based Biometric Person Authentication*, pages 254–259, 2004.
19. G. Guo, S.Z. Li, and K. Chan. Face recognition by Support Vector Machines. In *Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition*, pages 196–201, 2000.
20. Z.M. Hafed and M.D. Levine. Face recognition using the Discrete Cosine Transform. *International Journal of Computer Vision*, 43(3):167–188, 2004.
21. R.M. Haralick and L.G. Shapiro. *Computer and Robot Vision*. Prentice-Hall, 2002.
22. B. Heisele, H. Purdy, J. Wu, and T. Poggio. Face recognition: component-based versus global approaches. *Computer Vision and Image Understanding*, 91(1-2):6–21, 2003.
23. E. Hjelmås and B.K. Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83:236–274, 2001.
24. A.K. Jain, R. Bolle, and S. Pankanti, editors. *Biometrics - Personal Identification in Networked Society*. Kluwer, 1999.
25. D.J. Jobson, Z. Rahman, and G.A. Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, 6(3):451–462, 1997.
26. D.J. Jobson, Z. Rahman, and G.A. Woodell. Properties and performance of a center/surround retinex. *IEEE Transactions on Image Processing*, 6(3):451–462, 1997.
27. K. Jonsson, J. Kittler, Y.P. Li, and J. Matas. Support vector machines for face authentication. *Image and Vision Computing*, 2002.
28. T. Kanade. *Computer recognition of human faces*. Birkhauser, 1973.
29. M.D. Kelly. Visual identification of people by computer. Technical Report AI-130, Stanford University, 1970.
30. M. Kirby and L. Sirovich. Application of the karhunen-loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.
31. C. Kotropoulos and I. Pitas. Rule based face detection in frontal views. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2537–2540, 1997.
32. S.Y. Kung, M.W. Mak, and S.H. Lin. *Biometric Authentication - a Machine Learning Approach*. Prentice-Hall, 2005.
33. I.C. Kyong, K.W. Bowyer, and P.J. Flynn. Multiple-nose region matching for 3D face recognition under varying facial expression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1695–1700, 2006.
34. M. Lades, J. Vorbruggen, J. Buhmann, J. Lange, C.V.D. Malburg, and R. Wurtz. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computing*, 2:300–311, 1993.

35. E.H. Land and J.J. McCann. Lightness and retinex theory. *Journal of the Optical Society of America*, 61:1–11, 1971.
36. T.K. Leung, M.C. Burl, and P. Perona. Probabilistic affine invariants for recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 678–684, 1998.
37. C. Liu and H. Wechsler. Evolutionary pursuit and its application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:570–582, 2000.
38. J. Lu, K.N. Plataniotis, and A.N. Venetsanopoulos. Face recognition using kernel discriminant analysis algorithms. *IEEE Transactions on Neural Networks*, 14(1):117–126, 2003.
39. J. Lu, K.N. Plataniotis, and A.N. Venetsanopoulos. Face recognition using LDA-based algorithms. *IEEE Transactions on Neural Networks*, 14(1):195–200, 2003.
40. S. Marcel and S. Bengio. Improving face verification using skin color information. In *Proceedings of International Conference on Pattern Recognition*, 2002.
41. A.M. Martinez. Recognizing imprecisely localized, partially occluded, and expression variant faces from a single sample per class. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6):748–763, 2002.
42. A.M. Martinez and A.C. Kak. PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):228–233, 2001.
43. B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:696–710, 1997.
44. H. Moon and P.J. Phillips. Computational and performance aspects of PCA-based face recognition algorithms. *Perception*, 30:303–321, 2001.
45. E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 130–136, 1997.
46. C. Padgett and G. Cottrell. Representing face images for emotion classification. In *Advances in Neural Information Processing Systems*, 1997.
47. P. Penev and Atick. Local feature analysis: a general statistical theory for object representation. *Network: Computation in Neural Systems*, 7(3):477–500, 1996.
48. A. Pentland, B. Moghaddam, and T. Starner. View based and modular eigenspaces for face recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 84–91, 1994.
49. P.J. Phillips, R.M. McCabe, and R. Chellappa. Biometric image processing and recognition. In *Proceedings of European Conference on Signal Processing*, 1998.
50. P.J. Phillips, H. Moon, S.A. Rizvi, and P.J. Rauss. The FERET evaluation methodology for face recognition algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1090–1104, 2000.
51. P.J. Phillips, H. Wechsler, J. Huang, and P. Rauss. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing*, 16(5):296–305, 1998.
52. Z. Rahman, G. Woodell, and D. Jobson. A comparison of the multiscale retinex with other image enhancement techniques. In *Proceedings of IS&T 50th Anniversary Conference*, pages 19–23, 1997.
53. S.A. Rizvi, P.J. Phillips, and H. Moon. A verification protocol and statistical performance analysis for face recognition algorithms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 833–838, 1998.

54. Y. Rodriguez, F. Cardinaux, S. Bengio, and J. Mariéthoz. Estimating the quality of face localization for face verification. In *Proceedings of International Conference on Image Processing*, 2004.
55. D.L. Ruderman. The statistics of natural images. *Network: Computation in Neural Systems*, 5(4):598–605, 1994.
56. F. Samaria and S. Young. HMM based architecture for face identification. *Image and Vision Computing*, 3(1):71–86, 1991.
57. C. Samir, A. Srivastava, and M. Daoudi. Three-dimensional face recognition using shapes of facial curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1858–1863, 2006.
58. C. Sanderson and K.K. Paliwal. Polynomial features for robust face authentication. In *Proceedings of International Conference on Image Processing*, 2002.
59. C. Sanderson and K.K. Paliwal. Fast features for face authentication under illumination direction changes. *Pattern Recognition Letters*, 24:2409–2419, 2003.
60. J. Short, J. Kittler, and J. Messer. A comparison of photometric normalization algorithms for face verification. In *Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition*, pages 254–259, 2004.
61. L. Sirovich and M. Kirby. Low dimensional procedure for the characterization of human face. *Journal of Optical Society of America*, 4(3):519–525, 1987.
62. K.-K. Sung and T. Poggio. Example-based learning for view based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998.
63. D.L. Swets and J. Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Image Processing*, 18:831–836, 1996.
64. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.
65. M.A. Turk and A. Pentland. Face recognition using eigenfaces. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3–6, 1991.
66. J. Wayman, A.K. Jain, D. Maltoni, and D. Maio, editors. *Biometric Systems*. Springer-Verlag, 2005.
67. J. Wilder. Face recognition using transform coding of grayscale projections and the neural tree network. In R.J. Mammone, editor, *Artificial Neural Networks with Applications in Speech and Vision*, pages 520–536. Chapman Hall, 1994.
68. L. Wiskott, J.-M. Fellous, N. Kruger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):775–779, 1997.
69. H.-M. Yang. Face recognition using kernel methods. In *Advances in Neural Information Processing Systems*, 2002.
70. M.H. Yang and T.S. Huang. Human face detection in complex background. *Pattern Recognition*, 27(1):53–63, 1994.
71. M.H. Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, 2002.
72. W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458, 2003.
73. W. Zhao, A. Krishnaswamy, R. Chellappa, D. Swets, and J. Weng. Discriminant analysis of principal components for face recognition. In P.J. Phillips, V. Bruce, F.F. Soulie, and T.S. Huang, editors, *Face Recognition: from Theory to Applications*, pages 73–85. Springer-Verlag, 1998.



74. S. Zhou and R. Chellappa. Multiple exemplar discriminant analysis for face recognition. In *Proceedings of International Conference on Pattern Recognition*, pages 191–194, 2004.
75. S. Zhou, R. Chellappa, and B. Moghaddam. Intra-personal kernel space for face recognition. In *Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition*, pages 235–240, 2004.



## Video Segmentation and Keyframe Extraction

### What the reader should know to understand this chapter

- Basic notions of image processing (Chapter 3).
- Clustering techniques (Chapter 6).

### What the reader should know after reading this chapter

- State-of-the-art in video segmentation and shot detection.
- Feature extraction techniques for images.
- Performance measures for video segmentation systems.

### 14.1 Introduction

The goal of this chapter is to show how clustering techniques are applied to perform *video segmentation*, i.e. to split videos into segments meaningful from a semantic point of view. The segmentation is the first step of any process aimed at extracting from videos *high level* information, i.e. information which is not explicitly stated in the data, but it rather requires an abstraction process [10][17][22]. The video segmentation can be thought of as the partitioning of a text into chapters, sections and other parts that help the reader to better access the content. In more general terms, the segmentation of a long document (text, video, audio, etc.) into smaller parts addresses the limits of the human mind in dealing with large amounts of information. In fact, humans are known to be more effective when managing five to nine *information chunks* rather than a single *information block* corresponding to the sum of the chunks [30].

The general structure of a video is shown in Figure 14.1: the highest level segments are the *scenes* or *stories*, i.e. parts which are semantically coherent from the point of view of subject, people involved, etc. The intermediate layer are the *shots*, i.e. unbroken sequences of frames taken by one camera. The

transition between one shot and the following can be abrupt or gradual. In the first case the transition is called *cut*, in the second case is called *fade* or *dissolve*. At the lowest level there are the *keyframes*, i.e. the frames supposed to best represent the shot content. The vertical dotted line of Figure 14.1 separates *logical* and *physical* layers. Scenes and stories are said to form the logical layer because they are not characterized by physical properties, but rather by the author view, i.e. by the way the author organizes the video [40]. On the contrary, shots and keyframes are characterized by physical properties that enable one to extract them automatically. This is the reason why this chapter focuses on shot boundary detection and keyframe extraction.

The main problem in shot segmentation is to detect correctly the boundaries between one shot and the following one. This is done by applying two major approaches: the first estimates the difference between consecutive frames and identifies shot transitions as the points where such difference exceeds some threshold. The second is to apply clustering techniques to feature vectors extracted from single frames and to group into a shot all frames that tend to cluster together. Shot boundaries can be identified objectively and this enables one to have quantitative performance measures for automatic shot detection systems. The same does not apply to keyframe extraction because the most representative frame of a shot can be identified only on a subjective basis. However, it is still possible to ask human assessors to evaluate keyframe extraction systems and to provide judgments like in the case of the MOS score described in Chapter 2.

The main applications of video segmentation are digital libraries, video on demand, video browsing, video indexing and retrieval, etc. (see [40] for a survey) and in general all applications involving large collections of video recordings (see Section 14.2 for a quick survey). Video segmentation is the subject of both monographies [18][26][39] and surveys [10][17][22] that the interested reader can consult for more extensive information.

The rest of this chapter is organized as follows: Section 14.2 proposes a survey of major applications involving shot boundary detection and keyframe extraction. Section 14.3 presents the most common approaches to the problem of shot segmentation. Section 14.4 shows how to develop a simple shot boundary detection system using software packages available on the web. Section 14.5 describes keyframe extraction techniques. Section 14.6 shows how to create a simple keyframe extraction system using free software packages.

## 14.2 Applications of Video Segmentation

This section presents a survey of the major applications involving shot boundary detection and keyframe extraction. One of the most important domains where such tasks are performed is video indexing, i.e. the conversion of videos into a format suitable for retrieval systems. In such a context, the approach is typically as follows: first videos are segmented into shots, then



**Fig. 14.1.** Video structure. The picture shows the main components of a video.

keyframes are extracted from each shot. At this point, the problem of searching through videos can be performed through image retrieval, i.e. by retrieving keyframes similar to an image submitted as query. Such an approach is

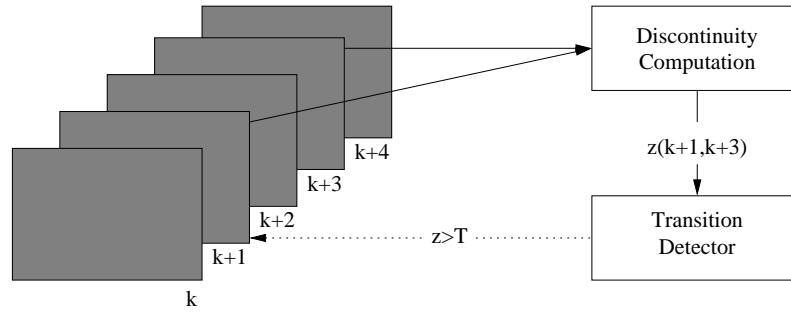
followed in [6][33][38][41][52] and, with some variants, in [19][23][25]: in [19] the authors try to group shots based on their content rather than to focus on single shots, in [23][50] the segmentation is performed hierarchically and in [9][25][49][54] hidden Markov models (see Chapter 10) are used to keep into account temporal constraints in an expected sequence of shots, in [28] the authors evaluate the impact of relevance feedback on the retrieval performance. Mathematical models based on video production techniques are proposed in [16][51][53].

Another important domain of application is *video browsing*, i.e. any technique trying to show the whole content of a video in a form as concise and accessible as possible. The need behind such an application is that users are often interested to watch only part of the video, e.g. the goals in a soccer match, and it should be possible to find such segment without watching the whole video. The most common approach is to segment approximately into scenes (see Figure 14.1) and then present the video as a sequence of keyframes. In this way few images can summarize several minutes. The users can then select the keyframes in order to access the corresponding video segments. Such an approach is used in [4][11][13][44][48]. The work in [4] tries to simplify the interaction of the users with the videos, while particular emphasis on using as less images as possible to represent a given video is placed in [48]. The works in [1][3][11][12][24][44] try to select the shots to be shown to the users based on content analysis rather than on simple physical properties. The use of unsupervised approaches to identify content coherent segments (and respective keyframes) is illustrated in [13][21][31]. The effects of real-time constraints on shot segmentation for browsing purposes are presented in [42].

Most of the applications aiming at analyzing the video content, i.e. what are the informations displayed in the video, use shots and keyframes as elementary units of information [8][43]: in [29] the authors use content analysis for coding purposes, in [45] the focus is on the use of compressed data to perform rapid scene analysis, and in [14][20][46][47] unsupervised techniques are used to explore the content of video collections. The application of principal component analysis (see Chapter 11) in such a domain is illustrated in [36]. Other applications are summarization [2][5], object detection [27], commercials detection [15][35], place identification [37] and classification of edit effects and motion [32][34].

### 14.3 Shot Boundary Detection

This section presents the main techniques for shot boundary detection in videos. The general scheme of a system is shown in Figure 14.2: given a sequence of frames  $f = \{f_1, \dots, f_F\}$ , the system computes a discontinuity function  $z(k, k + L)$  at each point  $k$ . The function  $z(k, k + L)$  measures the difference between frames  $k$  and  $k + L$  in the sequence. The parameter  $L$  is an offset and it must be set a priori. The value of  $L$  must be a tradeoff between



**Fig. 14.2.** Shot boundary detection system. The figure shows the general scheme of a shot boundary detection system: the frames are used to compute  $z(k, k + L)$ , in this case  $L = 2$ , and the shot detector inserts a boundary if  $z(k, k + L) > T$ .

two conflicting needs: the first is the detection of abrupt changes that can be identified by simply comparing two consecutive frames, i.e.  $L = 1$ ; the second is the detection of smooth transitions where the change is evident only when comparing frames at a certain distance from each other, i.e.  $L > 1$ . Whenever  $z(k, k + L) > T$ , where  $T$  is a predefined threshold, the system inserts a shot boundary.

The main problem of such an approach is that the parameters  $L$  and  $T$  must be set empirically and there is no guarantee that the two major criteria for robustness are met [17]:

- satisfactory detection performance across as many different videos as possible (this aspect depends critically on  $T$ ).
- satisfactory detection performance for both hard and gradual transitions (this aspect depends critically on  $L$ )

The next sections present some of the most commonly applied approaches used to compute  $z(k, k + l)$  and the performance measures used to assess the effectiveness of shot boundary detection systems.

### 14.3.1 Pixel-Based Approaches

The simplest form of discontinuity function is the pixel-by-pixel difference between frames  $k$  and  $k + L$ . In the case of gray-level images, the expression of the difference is:

$$z(k, k + L) = \frac{\sum_{i=1}^R \sum_{j=1}^C |I_k(i, j) - I_{k+L}(i, j)|}{CR} \quad (14.1)$$

where  $I_k(i, j)$  is the intensity level of pixel  $(i, j)$  in frame  $k$ ,  $C$  is the number of columns per frame, and  $R$  is the number of rows per frame. When the frames are color images, then the value of each pixel corresponds to a triple  $c = (c_1, c_2, c_3)$  and the difference function becomes:

$$z(k, k + L) = \frac{\sum_{i=1}^R \sum_{j=1}^C \sum_{c=1}^3 |I_k(i, j, c) - I_{k+L}(i, j, c)|}{CR}, \quad (14.2)$$

where  $I_k(i, j, c)$  is the value of the  $c$  component of pixel  $(i, j)$  in frame  $k$  (see Chapter 3 for the meaning of  $c_i$  components).

Pixel comparison is often used as a baseline for comparison with more complex approaches, but it has a major drawback, i.e. it cannot distinguish between a small change in a large area and a large change in a small area. As an example consider two frames such that  $I_{k+L}(i, j) = I_k(i, j) + 1$ . The two images are visually similar and they are unlikely to correspond to a shot transition. The value of  $z(k, k + L)$  for the frames of the above example is 1 and it can be obtained also for two images where the changes are concentrated in a small area:

$$\begin{cases} I_{k+L}(i, j) = I_k(i, j) + CR/P^2 & i, j \leq P \\ I_{k+L}(i, j) = I_k(i, j) & i, j > P \end{cases} \quad (14.3)$$

where  $P$  is an arbitrary constant such that  $P \leq C$  and  $P \leq R$ . The above images are visually different and they are likely to correspond to a shot transition, but the value of the difference is the same as in the first of the above examples where the images were not likely to account for a shot boundary.

The above effect is reduced by introducing the following function (the extension to color images is straightforward):

$$DP(k, k + L, i, j) = \begin{cases} 1 & \text{if } |I_k(i, j) - I_{k+L}(i, j)| > T_2 \\ 0 & \text{otherwise} \end{cases} \quad (14.4)$$

where  $T_2$  is an arbitrary threshold. The difference between two frames can then be computed as follows:

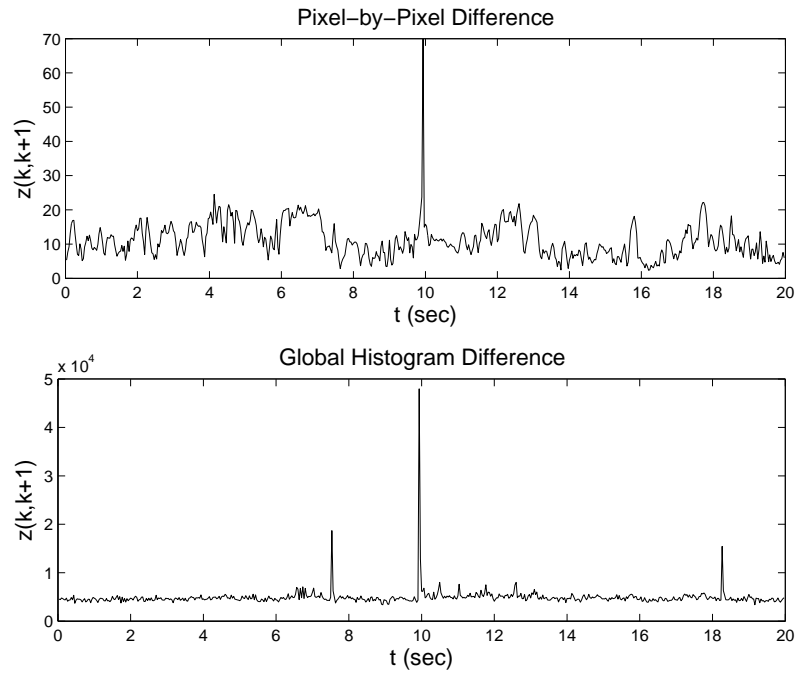
$$z(k, k + L) = \frac{\sum_{i=1}^R \sum_{j=1}^C DP(k, k + L, i, j)}{CR} \quad (14.5)$$

(the extension to color images is straightforward). This function addresses the problem described above, i.e. the distinction between large changes concentrated in small areas and small changes diffuse over large areas, but leaves open another problem: when the camera captures a subject moving slowly from left to right, the number of pixels where the threshold  $T_2$  is exceeded is high, but this does not correspond to a shot transition. The problem is typically avoided by replacing the value of each pixel with the average of the surrounding pixels.

### 14.3.2 Block-Based Approaches

Pixel-based approaches are sensitive to moving objects, i.e. they tend to insert shot transitions where the images show an object moving with respect to a fixed background. In general this is not correct because the frame sequence





**Fig. 14.3.** Discontinuity functions. The upper plot shows the pixel-based difference, the lower plot shows the global histogram-based difference.

is unbroken (see the definition of *shot* in Section 14.1) even if the object changes position. Block-based approaches deal with such a problem by using discontinuity functions of the following form:

$$z(k, k + L) = \sum_{n=1}^B D(k, k + L, n) \quad (14.6)$$

where  $B$  is the number of blocks, and  $D(k, k + L, n)$  is the difference between block  $n$  in frames  $k$  and  $k + L$ . A block is a subset of the image, e.g. a square with lower left corner in pixel  $(i, j)$  and upper right corner in pixel  $(i + N, j + N)$ . The difference function  $D(\cdot)$  is typically one of the pixel-based expressions described in the previous section. In some cases, the blocks cover the whole image (eventually overlapping each other), while in other cases they are arranged over a grid supposed to cover the most important areas of the images.

Block-based approaches are more robust than pixel-based approaches to moving objects or to slow movements of the camera, but they are slower because they require more computation.

### 14.3.3 Histogram-Based Approaches

Both pixel- and block-based approaches are affected by the spatial disposition of gray levels (or colors) in the frame. This is the reason why objects changing position or camera movements create problems. Histogram-based approaches deal with such an effect because they use information related to the distribution of pixel values without taking into account their position.

If  $N$  gray-level values  $1, 2, \dots, N$  are possible, the histogram  $\mathbf{H}_k$  of frame  $k$  is an  $N$ -dimensional vector where component  $H_k(i)$  accounts for the number of occurrences (or for the percentage) of pixels where the gray level is  $i$ . In the case of color images, the number of possible colors is  $N^3$  and the histogram has the same number of components. The simplest discontinuity function based on histograms is the following:

$$z(k, k + L) = \sum_{i=1}^N |H_k(i) - H_{k+L}(i)| \quad (14.7)$$

and simply corresponds to the difference between vectors  $\mathbf{H}_k$  and  $\mathbf{H}_{k+L}$ .

Similar approaches try to enhance the difference between histograms extracted in different shots by using a  $\chi^2$  variable as discontinuity function:

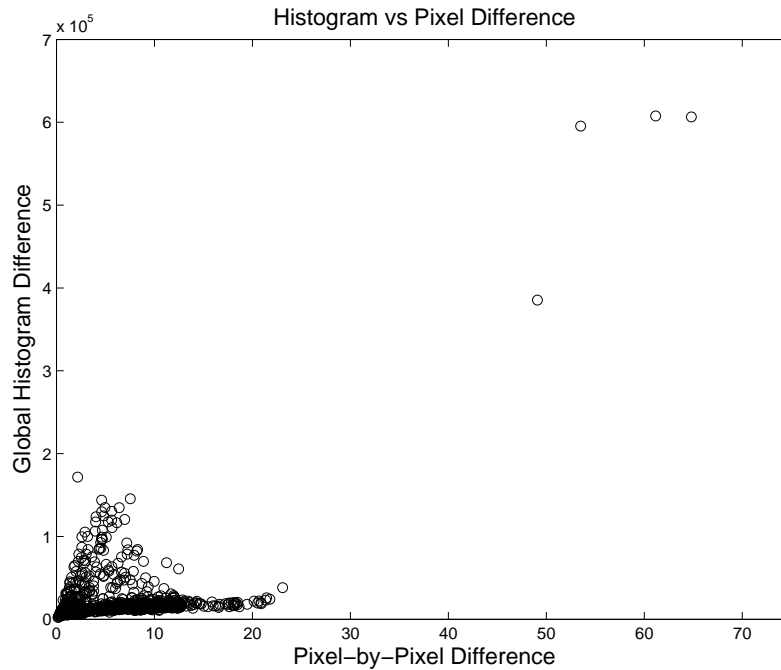
$$z(k, k + L) = \sum_{i=1}^N \frac{|H_k(i) - H_{k+L}(i)|^2}{H_{k+L}(i)}. \quad (14.8)$$

However, the above function tends also to enhance differences between frames of the same shot and this results in the insertion of false transitions. The same discontinuity functions can be applied to image blocks such as in the case of pixel-based techniques (see above).

### 14.3.4 Clustering-Based Approaches

The approaches described so far have two major drawbacks: the first is that they require us to set empirically one or more thresholds. This is a problem because threshold values are often data dependent and the algorithms fail in matching the first condition stated at the beginning of this section, i.e. that the performance of the system should be uniform across different kinds of data and require minimum effort in parameter tuning when changing data. The second is that each algorithm is adapted to a specific problem, but fails in addressing the others: e.g. histogram-based approaches address the problem of moving objects (see above), but have problems in detecting gradual transitions. In other words, the above approaches do not meet the second condition stated at the beginning of the section, i.e. that the algorithms must be capable of detecting with satisfactory performance all kinds of transitions.

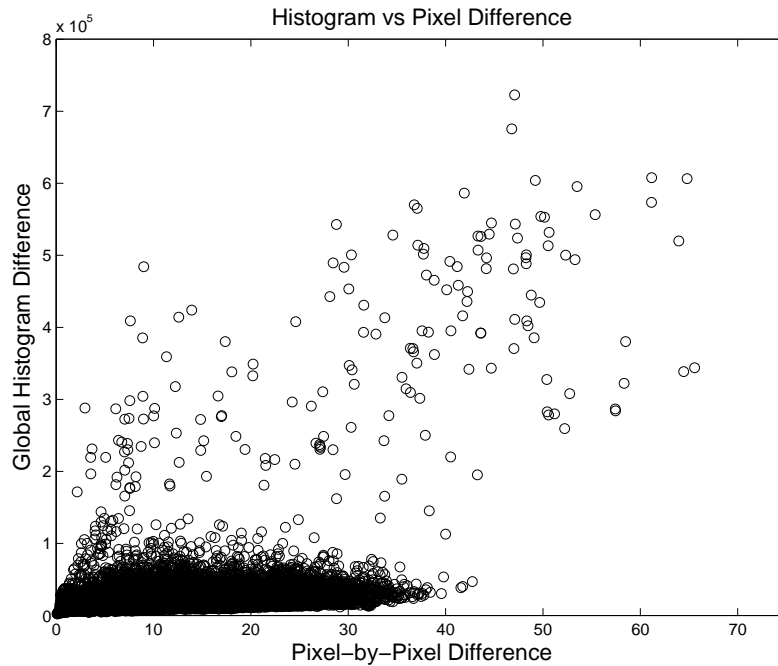
Clustering-based approaches try to address the above limits by applying unsupervised learning algorithms to frame changes. The reason is that clustering algorithms do not require us to set thresholds and are expected to



**Fig. 14.4.** Histogram vs. pixel difference for a single video. Each point in the plot corresponds to a pair of consecutive frames in a 60-seconds long video. The horizontal axis is the pixel-by-pixel difference and the vertical axis is the global histogram axis. The points close to the upper right corner of the plot are the actual shot transitions of the video.

group all frame changes into two classes: shot transitions and others. The second is that clustering algorithms enable us to use multiple features capable of addressing at the same time the different problems presented above. This concept is illustrated in Figure 14.4 where the horizontal axis is the pixel-based difference and the vertical axis is the global histogram-based difference. Each point corresponds to a pair of consecutive frames in the video used in Figure 14.1. The four points close to the upper right corner correspond to the actual shot boundaries of the video.<sup>1</sup> The separation between *ordinary* frames and shot boundaries is evident. In this video, the two classes are even linearly separable, but the problem is more difficult when the plots are obtained using large databases of videos. In fact, the variability of the data tend to form different clusters corresponding to different kinds of transitions: sometimes the last image of a shot is very different from the first image of the following

<sup>1</sup> Figure 14.4 includes the whole video and this is the reason why the shot boundaries are four rather than one as shown in Figure 14.1 which shows only the first 20 seconds.



**Fig. 14.5.** Histogram vs. pixel difference for several videos (see Figure 14.4). Normal frames tend to concentrate in the lower left corner (where the two differences are close to zero), while shot boundaries tend to cluster towards the upper right corner (where the two differences are significantly higher than zero).

shot and the boundary is clear; other times, the new shot is simply a different view of the same subject and the transition is more difficult to capture. Such a situation is evident in Figure 14.5 where the points are extracted from a set of 15 videos. However, clustering-based approaches have been shown through extensive experiments to be more robust than other methods.

### 14.3.5 Performance Measures

The performance of a shot boundary detection system is measured in terms of *precision*  $\pi$  and *recall*  $\rho$ : the first is the fraction of frames identified as shot boundaries by the system that correspond to real shot boundaries; the second is the percentage of real shot boundaries that have been detected as such by the system. In mathematical terms, if  $R_d$  is the set of frames that the system claims to be shot boundaries and  $R_t$  is the set of frames that correspond to real shot boundaries, then the precision is defined as follows:

$$\pi = \frac{|R_d \cap R_t|}{|R_d|}, \quad (14.9)$$

where  $|\cdot|$  is the cardinality of the set, and the recall is:

$$\rho = \frac{|R_d \cap R_t|}{|R_t|}. \quad (14.10)$$

The above expressions show that precision and recall can be interpreted as probabilities:  $\pi$  is the probability that a frame identified as a boundary by the system is actually a boundary, and  $\rho$  is the probability that a boundary is identified as such by the system.

precision and recall are not independent and must always be used together. Consider as an example a video with  $N$  shot boundaries and  $F$  frames. If all frames are classified as shot boundaries, then  $\rho = 100\%$  and the system seems to work correctly, but  $\pi = N/F$ , i.e. the smallest possible value for the precision given  $N$  and  $F$ . Conversely, a system that classifies as shot boundary only the frame with the highest difference with respect to its following frame (see above) would probably have  $\pi = 100\%$  because such a frame is likely to be a shot boundary, but the recall would be  $1/N$ , i.e. the smallest possible value given  $N$  and  $F$ . Both above examples show that using only  $\rho$  or only  $\pi$  is potentially misleading.

## 14.4 Shot Boundary Detection with *Torchvision*

This section shows how a simple shot boundary detection system can be implemented using *Torchvision*,<sup>2</sup> the software package presented in Chapter 13. Given a video, the first problem is to extract the frame it contains and this can be done using the following function:

```
avi2avippm -ppm -ffmpeg -I num video.avi
```

where `video.avi` is the video to be analyzed (in AVI format) and the effect of the options is as follows:

- `ppm` specifies the format of the image files where the frames are stored (only the ppm format is currently available).
- `ffmpeg` specifies that the encoder is *mpeg*.
- `I` specifies the number of frames to be extracted (if no value is specified, the program extracts all the frames in the video).

The above program gives as output an image for each frame of the video. In general there are 24 frames per second, then even a short video results in several thousands of pictures. The images are numbered following the frame order and this enables one to calculate the differences between frames  $k$  and  $k+L$  using the algorithms described in Section 14.3. The *Torchvision* package contains a function that performs the pixel-by-pixel difference:

<sup>2</sup> At the time this book is being written, the package can be downloaded from <http://torch3vision.idiap.ch>.

```
imagediff frame_k.ppm frame_l.ppm
```

where `frame_k.ppm` and `frame_l.ppm` are the two images to be compared. Once the parameter  $L$  has been set, the `imagediff` function enables one to obtain a difference value at each instant and to plot a curve like the one of Figure 14.3.

Alternatively, *Torchvision* proposes also a function for collecting the image histograms from gray-level images:

```
histopgm frame.pgm
```

where `frame.pgm` is the image file containing a frame in pgm format.

The joint use of both `imagediff` and `histopgm` enables one to obtain scatter plots like those in Figure 14.4 and 14.5. Such data can be clustered using the techniques and the packages presented in Chapter 6.

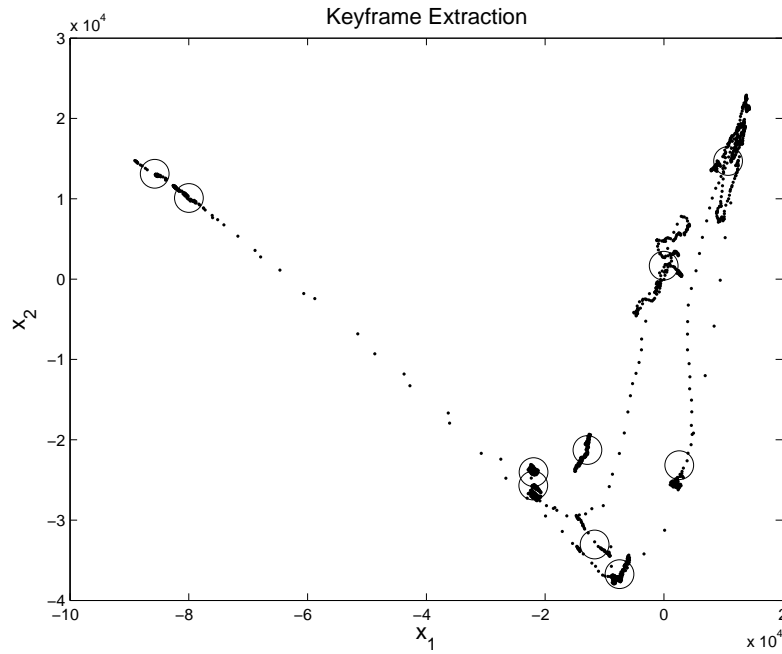
## 14.5 Keyframe Extraction

The segmentation into shots is the first step of many video processing applications (see Section 14.2). However, shots are still difficult to handle. They are often replaced with one of their frames supposed to be *representative* of their content. By representative it is meant here that the application of any algorithm to such a frame leads to the same results that would be obtained by applying the same algorithm to the whole shot.

The most common approach to the keyframe extraction problem is to extract a feature vector from each frame and then to apply a clustering algorithm to the resulting data. After, the keyframe is identified as the frame closest to the centroid of the largest or of the smallest cluster: in the first case, the rationale is to represent the shot with the frame showing the most frequent characteristics, in the second case the rationale is to represent the shot with the rarest characteristics. Since there are no metrics accounting for the keyframe extraction process, none of the above approaches can be proposed as better than the other. In general, both techniques lead to reasonable results and allow one to perform further processing steps such as indexing, retrieval, browsing, etc.

Another typical approach is to avoid the segmentation into shots and to cluster the whole set of frames extracted from a given video. In this case, the frames closest to the cluster centroids are expected to be representative of the video content because they are at the center of densest regions, i.e. they show characteristics common to many different frames.

An example of such a technique is shown in Figure 14.6 where each point corresponds to a frame in a 75 seconds long video and the circles are centered around the centroids found by applying the K-means algorithm presented in Chapter 6. The features have been obtained as follows: the histogram has been extracted from each frame and principal component analysis (see Chapter 11) has been applied to the resulting vectors. The features  $x_1$  and  $x_2$  (the



**Fig. 14.6.** Frame clustering. Each point corresponds to a frame and each circle corresponds to a centroid obtained with the K-means algorithm.

horizontal and vertical axis of Figure 14.6 respectively) are the projections of the histogram vectors onto the first two principal components.

The frames belonging to each shot tend to cluster because they are visually similar and their histograms are thus close to each other. The transitions between neighboring shots are gradual and this results into the points connecting different clusters in a *filament-like* structure. The original dimension of the histograms is 256, but the use of just two features leads to satisfactory results. The reason is that many frames are similar to one another (cameras capture 24 frames per second and no major changes happen at such a time-scale), then there is a large amount of redundancy. Section 14.6 shows how to apply the above approach using the *Torch* package.

In the approaches described so far, no temporal constraints are taken into account, i.e. the order of the frames along the video is not used. In some cases, such an information is useful because the visual difference between temporally close frames can be due to local effects such as illumination changes or moving objects. In this case, the frames should cluster together despite the visual differences.

## 14.6 Keyframe Extraction with *Torchvision* and *Torch*

This section shows how to perform keyframe extraction by using the *Torch* package [7], an extensive software library including the most common techniques applied in machine learning <sup>3</sup>.

The first step of the process is the conversion of the video into a sequence of images and it can be performed using the function `avi2avippm` described in Section 14.4. The following step is the extraction of the histograms from the single frame images and it can be performed using the function `histopgm` described in Section 14.4. The PCA can be extracted using the function `trainPca` of *TorchVision* as explained in Chapter 13.

The last step is the application of the K-means algorithm to the projections of the histogram algorithms onto the first  $N$  principal components. An implementation of the K-means is available in *Torch* and it can be called as follows:

```
kmeans -save model -one_file data.dat
```

where the meaning of the options is as follows:

- `-save` specifies the file where the centroid coordinates must be stored (`model` in the example).
- `one_file` specifies that all training examples are in a single file (called `data.dat` in the example).

Once the  $K$  centroids are available, is up to the user to decide whether to select as keyframes the images closest to largest or smallest clusters.

This section proposes the K-means because it is the simplest clustering algorithm and it represents a good baseline. However, any other clustering algorithm can be used for the same application. Chapter 6 presents a large variety of clustering algorithms including available software packages that implement them.

---

<sup>3</sup> At the time this book is being written, the package is publicly available at the following website: <http://www.torch.ch>.



---

## References

1. M. Abdel-Mottaleb, N. Dimitrova, R. Desai, and J. Martino. CONIVAS: content based image and video access system. In *Proceedings of ACM International Conference on Multimedia*, pages 427–428, 1996.
2. A. Aner-Wolf and J. Kender. Video-summaries and cross-referencing through mosaic based representation. *Computer Vision and Image Understanding*, 95(2):201–237, 2004.
3. H. Aoki, S. Shmotsuji, and O. Hori. A shot classification method of selecting effective key-frames for video browsing. In *Proceedings of ACM International Conference on Multimedia*, pages 1–10, 1996.
4. R. Castagno, T. Ebrahimi, and M. Kunt. Video segmentation based on multiple features for interactive multimedia applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):562–571, 1998.
5. Z. Cernekova, I. Pitas, and Nikou. Information theory-based shot cut/fade detection and video summarization. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(1):82–91, 2006.
6. H.S. Chang, S. Sull, and S.U. Lee. Efficient video indexing scheme for content-based retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1269–1279, 1999.
7. R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical Report 02-46, IDIAP, 2002.
8. P.L. Correia and F. Pereira. Classification of video segmentation application scenarios. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(5):735–741, 2004.
9. J.M. Corridoni and A. Del Bimbo. Structured representation and automatic indexing of movie information content. *Pattern Recognition*, 31(12):2027–2045, 1998.
10. N. Dimitrova, H.J. Zhang, B. Shahraray, I. Sezan, T. Huang, and A. Zakhor. Application of video-content analysis and retrieval. *IEEE Multimedia*, 9(3):42–55, 2002.
11. A.D. Doulamis and N.D. Doulamis. Optimal content-based video decomposition for interactive video navigation. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(6):757–775, 2004.

12. X. Du and G. Fan. Joint key-frame extraction and object segmentation for content-based video analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(7):904–914, 2006.
13. X. Gao and X. Tang. Unsupervised video-shot segmentation and model-free anchorperson detection for news video story parsing. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(9):765–776, 2002.
14. D. Gatica-Perez, A. Loui, and M.T. Sun. Finding structure in home videos by probabilistic hierarchical clustering. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(6):539–548, 2003.
15. J.M. Gauch and A. Shivadas. Finding and identifying unknown commercials using repeated video sequence detection. *Computer Vision and Image Understanding*, 103(1):80–88, 2006.
16. A. Hamampur, T. Weymouth, and R. Jain. Digital video segmentation. In *Proceedings of ACM International Conference on Multimedia*, pages 357–364, 1994.
17. A. Hanjalic. Shot boundary detection: unraveled and resolved? *IEEE Transactions on Circuits and Systems for Video Technology*, 12(2):90–105, 2002.
18. A. Hanjalic. *Content Based Analysis of Digital Video*. Springer-Verlag, 2004.
19. A. Hanjalic, R.L. Lagendijk, and J. Biemond. Automated high-level movie segmentation for advanced video-retrieval systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(4):580–588, 1999.
20. A. Hanjalic and H.J. Zhang. An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1280–1289, 1999.
21. V. Kobla, D. Doermann, and C. Faloutsos. VideoTrails: representing and visualizing structure. In *Proceedings of ACM International Conference on Multimedia*, pages 335–346, 1997.
22. I. Koprinska and S. Carrato. Temporal video segmentation: a survey. *Signal Processing: Image Communication*, 16:477–500, 2001.
23. J. Lee and B.W. Dickinson. Hierarchical video indexing and retrieval for subband-coded video. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(5):824–829, 2000.
24. M.S. Lee, Y.M. Yang, and S.W. Lee. Automatic video parsing using shot boundary detection and camera operation analysis. *Pattern Recognition*, 34(3):711–719, 2001.
25. R. Leonardi, P. Migliorati, and M. Prandini. Semantic indexing of soccer audiovisual sequences: a multimodal approach based on controlled Markov chains. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(5):634–643, 2004.
26. Y. Li and J. Kuo. *Video Content Analysis Using Multimodal Information*. Springer-Verlag, 2003.
27. L. Lije and G. Fan. Combined key-frame extraction and object-based video segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(7):869–884, 2005.
28. S.D. MacArthur, C.E. Brodley, A.C. Kak, and L.S. Broderick. Interactive content-based image retrieval using relevance feedback. *Computer Vision and Image Understanding*, 88(2):55–75, 2002.
29. T. Meier and K.N. Ngan. Video segmentation for content-based coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1190–1203, 1999.

30. G.A. Miller. The magic number seven plus or minus two: some limits on capacity for processing information. *Psychology Review*, 63:81–97, 1956.
31. C.W. Ngo, T.C. Pong, and R.T. Chin. Video partitioning by temporal slice coherency. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(8):941–953, 2001.
32. N.V. Patel and I.K. Sethi. Video shot detection and characterization for video databases. *Pattern Recognition*, 30(4):583–592, 1997.
33. M.J. Pickering and S. Rüger. Evaluation of key-frame based retrieval techniques for video. *Computer Vision and Image Understanding*, 92(2-3):217–235, 2003.
34. S. Porter, M. Mirmehdi, and B. Thoams. Temporal video segmentation and classification of edit effects. *Image and Vision Computing*, 21(13-14):1097–1106, 2003.
35. K.M. Pua, Gauch J.M., S.E. Gauch, and J.Z. Miadowicz. Real-time repeated video sequence identification. *Computer Vision and Image Understanding*, 93(3):310–327, 2004.
36. E. Sahouria and A. Zakhor. Content analysis of video using principal component analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1290–1298, 1999.
37. F. Schaffalitzky and A. Zisserman. Automated location matching in movies. *Computer Vision and Image Understanding*, 92(2-3):217–235, 2003.
38. M.A. Smith and M.G. Christel. Automating the creation of a digital video library. In *Proceedings of ACM International Conference on Multimedia*, pages 357–358, 1995.
39. M.A. Smith and T. Kanade. *Multimodal Video Characterization and Summarization*. Springer-Verlag, 2004.
40. C.G.M. Snoek and M. Worring. Multimodal video indexing: a review of the state-of-the-art. *Multimedia Tools and Applications*, 25(1):5–35, 2005.
41. K.W. Sze, K.M. Lam, and G. Qiu. A new key frame representation for video segment retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(9):1148–1155, 2005.
42. Y. Taniguchi, A. Akutsu, Y. Tonomura, and H. Hamada. An intuitive and efficient access interface to real-time incoming video based on automatic indexing. In *Proceedings of ACM International Conference on Multimedia*, pages 25–33, 1995.
43. B.T. Truong, S. Venkatesh, and C. Dorai. Scene extraction in motion pictures. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(1):5–15, 2003.
44. S. Tsekeridou and I. Pitas. Content-based video parsing and indexing based on audi-visual interaction. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(4):522–535, 2001.
45. D. Wang. Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):539–546, 1998.
46. B.L. Yeo and B. Liu. Rapid scene analysis on compressed video. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(6):533–544, 1995.
47. M. Yeung, B.L. Yeo, and B. Liu. Segmentation of video by clustering and graph analysis. *Computer Vision and Image Understanding*, 71(1):94–109, 1998.
48. M.M. Yeung and B.L. Yeo. Video visualization for compact presentation and fast browsing of pictorial content. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(5):771–785, 1997.

49. H. Yi, D. Rajan, and L.T. Chia. A motion-based scene tree for compressed video content management. *Image and Vision Computing*, 24(2):131–142, 2006.
50. H.H. Yu and W. Wolf. A hierarchical multiresolution video shot transition detection scheme. *Computer Vision and Image Understanding*, 75(1-2):196–213, 1999.
51. H.J. Zhang, C.Y. Low, S.W. Smoliar, and J.H. Wu. Video parsing, retrieval and browsing: an integrated and content-based solution. In *Proceedings of ACM International Conference on Multimedia*, pages 15–24, 1995.
52. H.J. Zhang, J. Wu, D. Zhong, and S.W. Smoliar. An integrated system for content-based video retrieval and browsing. *Pattern Recognition*, 30(4):643–658, 1997.
53. H.J. Zhang, J.H. Wu, C.Y. Low, and S.W. Smoliar. A video parsing, indexing and retrieval system. In *Proceedings of ACM International Conference on Multimedia*, pages 359–360, 1995.
54. Y.J. Zhang and H.B. Lu. A hierarchical organization scheme for video data. *Pattern Recognition*, 35(11):2381–2387, 2002.

Appendices



# A

---

## Statistics

### A.1 Fundamentals

This section provides the fundamentals of probability and statistics. The concept of probability of an event is introduced as a limit of the relative frequency, i.e. of the number of times an experiment has such an event as outcome. Based on such a definition, the rest of this section introduces the *addition law*, defines the *conditionality* and the *statistical independence*.

#### A.1.1 Probability and Relative Frequency

Consider the simple experiment of tossing an unbiased coin: two mutually exclusive outcomes are possible, head ( $H$ ) or tail ( $T$ ), and the result is *random*, i.e. it cannot be predicted with certainty because too many parameters should be taken into account to model the motion of the coin. On the other hand, if the experiment is repeated a sufficient number of times and a whole series of *independent trials under identical conditions* is obtained, the outcome shows some regularities: the fraction of experiments with outcome  $H$ , the so-called *relative frequency* of  $H$ , is always around  $1/2$ :

$$\frac{n(H)}{n} \simeq \frac{1}{2} \tag{A.1}$$

where  $n(H)$  is the number of times that the outcome is  $H$  and  $n$  is the total number of experiments. The same considerations apply to the  $T$  outcome and this is what the common language means when it says that the *probability* of  $H$  or  $T$  is 50 percent.

In more general terms, if an experiment has  $K$  mutually exclusive possible outcomes  $A_1, A_2, \dots, A_K$ , the probability  $p(A_i)$  of observing the outcome  $A_i$  can be thought of as the following limit:

$$p(A_i) = \lim_{n \rightarrow \infty} \frac{n(A_i)}{n} \tag{A.2}$$

(see above for the meaning of symbols). This result is known as the *strong law of large numbers* and it provides the definition of the probability.<sup>1</sup>

### A.1.2 The Sample Space

A random experiment is characterized by a set  $\Omega$  of *mutually exclusive* elementary events  $\omega$  that correspond to all its possible outcomes.  $\Omega$  is called a *sample space* and an event  $A$  is said to be *associated* with it when it is always possible to decide whether the occurrence of an elementary event  $\omega$  leads to the occurrence of  $A$  or not. As an example consider the rolling of a die; the sample space contains six elementary events  $\omega_1, \dots, \omega_6$  corresponding to the number of spots on each of the die faces. The event of having an even number of spots is associated to  $\Omega$  because it is a characteristic that can be clearly attributed to each of the elementary events, and it can be thought of as a set  $A = \{\omega_2, \omega_4, \omega_6\}$ . In the following,  $A$  will refer not only to an event, but also to the corresponding set of its underpinning elementary events and whenever there is no ambiguity, the distinction will not be made.

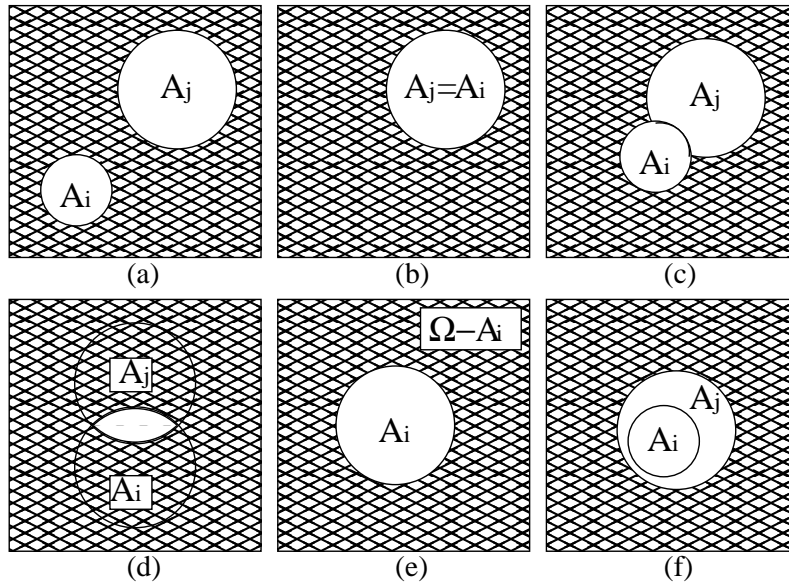
Based on the above, an event can be defined as a subset of the sample space and this enables to interpret the event properties and relationships in terms of sets and subsets as shown in Figure A.1. Two events  $A_i$  and  $A_j$  are said to be *mutually exclusive* when the occurrence of one prevents the other from occurring. This situation is shown in Figure A.1 (a) where the sets of elementary events corresponding to  $A_i$  and  $A_j$  are disjoint. When  $A_i$  and  $A_j$  contain exactly the same elements of the sample space, then the occurrence of one corresponds to the occurrence of the other and the two events are said to be *equivalent* (Figure A.1 (b)). The *union*  $A_i \cup A_j$  of two events is the event including all elements  $\omega$  of both  $A_i$  and  $A_j$ , while their *intersection*  $A_i \cap A_j$  contains only elementary events belonging to both  $A_i$  and  $A_j$ , as shown in Figure A.1 (c) and (d), respectively. Two events  $A_i$  and  $A_j$  are called *complementary* when  $A_i = \Omega - A_j = \bar{A}_j$  and the occurrence of one is equivalent to the nonoccurrence of the other. The difference between complementarity and mutual exclusivity is that  $\bar{A}_i$  contains all events of  $\Omega$  that are mutually exclusive with respect to  $A_i$ . On the other hand, complementarity and mutual exclusivity are the same property when there are only two events. The event  $A_i$  implies  $A_j$  when  $A_i \subset A_j$ , i.e. when the occurrence of  $A_i$  corresponds to the occurrence of  $A_j$ , but the vice versa is not true. This situation is depicted in Figure A.1 (f).

### A.1.3 The Addition Law

Consider two mutually exclusive events  $A_i$  and  $A_j$  and the event  $A = A_i \cup A_j$ . If both  $A_i$  and  $A_j$  belong to the sample space of an experiment repeated  $n$  of

<sup>1</sup> The *Strong law of large numbers* will not be demonstrated in this appendix. However, the interested reader can find the demonstration and related issues in most of the academic statistics textbooks





**Fig. A.1.** Relationships between events. This figure shows the different relationships between events in the sample space. Plot (a) shows mutually exclusivity, plot (b) shows equivalence, plot (c) and (d) correspond to union and intersection respectively, plot (e) shows the complementarity and plot (f) shows the inclusion.

times under identical conditions, then the relationship between the respective relative frequencies is as follows:

$$\frac{n(A)}{n} = \frac{n(A_i)}{n} + \frac{n(A_j)}{n}. \tag{A.3}$$

Section A.1.1 shows that the relative frequency tends to the probability when  $n \rightarrow \infty$ ; thus the above equation corresponds to:

$$P(A) = P(A_i) + P(A_j). \tag{A.4}$$

If the mutually exclusive events are  $k$ , then  $A = A_1 \cup A_2 \dots A_k$  and it is possible to write:

$$P(A) = P(A_1 \cup A_2 \dots A_{k-1}) + P(A_k) \tag{A.5}$$

and the above expression, after applying  $k - 2$  times Equation (A.4), leads to the *addition law for probabilities*:

$$P(A) = P\left(\bigcup_{l=1}^k A_l\right) = \sum_{l=1}^k P(A_l). \tag{A.6}$$

The above expression is valid only for mutually exclusive events, but an addition law can be obtained also for arbitrary events. This requires to demonstrate some key relationships between probabilities:

**Theorem A.1.** *The formulas*

$$0 \leq P(A) \leq 1 \quad (\text{A.7})$$

$$P(A_i - A_j) = P(A_i) - P(A_i \cap A_j) \quad (\text{A.8})$$

$$P(A_j - A_i) = P(A_j) - P(A_i \cap A_j) \quad (\text{A.9})$$

$$P(A_i \cup A_j) = P(A_i) + P(A_j) - P(A_i \cap A_j) \quad (\text{A.10})$$

where  $A_i - A_j$  stands for event  $A_i$  occurring without event  $A_j$  occurring as well, hold for arbitrary events  $A$ ,  $A_i$  and  $A_j$ . Moreover, if  $A_i \subseteq A_j$ , then:

$$P(A_i) \leq P(A_j). \quad (\text{A.11})$$

Equation (A.7) follows from the fact that the probability can be interpreted as a limit of the relative frequency  $n(A)/n$ . The value of  $n(A)$  is the number of times the experiment has  $A$  as outcome, thus it cannot be less than 0 and it cannot be more than  $n$ . As a consequence:

$$0 \leq \frac{n(A)}{n} \leq 1. \quad (\text{A.12})$$

Such relationships hold also when  $n \rightarrow \infty$  and this leads to Equation (A.7).

The events  $A_i$ ,  $A_j$  and  $A_i \cup A_j$  can be written as unions of mutually exclusive events as follows:

$$\begin{aligned} A_i &= (A_i - A_j) \cup (A_i \cap A_j) \\ A_j &= (A_j - A_i) \cup (A_i \cap A_j) \\ A_i \cup A_j &= (A_i - A_j) \cup (A_j - A_i) \cup (A_i \cap A_j) \end{aligned}$$

Since all events involved in the above equations are mutually exclusive, the application of the addition law leads to Equations (A.8), (A.9) and (A.10), respectively.

When  $A_i \subset A_j$ , the probability of  $A_j - A_i$  is:

$$P(A_j - A_i) = P(A_j) - P(A_i \cap A_j) = P(A_j) - P(A_i) \quad (\text{A.13})$$

because  $A_i \cap A_j = A_i$ . Since  $P(A_j - A_i) \geq 0$ ,

$$P(A_i) \leq P(A_j). \quad (\text{A.14})$$

which corresponds to Equation A.12.

After proving the relationships of Theorem A.1, it is possible to avoid the requirement of the mutual exclusivity for the addition law:

**Theorem A.2.** *Given any  $n$  events  $A_1, A_2, \dots, A_n$ , let*

$$P_1 = \sum_{i=1}^n P(A_i) \quad (\text{A.15})$$

$$P_2 = \sum_{1 \leq i < j \leq n} P(A_i A_j) \quad (\text{A.16})$$

$$P_3 = \sum_{1 \leq i < j < k \leq n} P(A_i A_j A_k) \dots \quad (\text{A.17})$$

where  $A_i A_j \dots A_k$  is a shorthand for  $A_i \cap A_j \dots \cap A_k$ , then:

$$P\left(\bigcup_{l=1}^n A_l\right) = P_1 - P_2 + P_3 + \dots + (-1)^{n+1} P_n. \quad (\text{A.18})$$

When  $n = 2$ , Equation (A.18) corresponds to Equation (A.10); then it is proved. Suppose now that (A.18) holds for  $n - 1$ ; then:

$$P\left(\bigcup_{l=2}^n A_l\right) = \sum_{i=2}^n P(A_i) - \sum_{2 \leq i < j \leq n} P(A_i A_j) + \dots \quad (\text{A.19})$$

and

$$P\left(\bigcup_{l=2}^n A_1 A_l\right) = \sum_{i=2}^n P(A_1 A_i) - \sum_{2 \leq i < j \leq n} P(A_1 A_i A_j) + \dots \quad (\text{A.20})$$

Based on Equation (A.10), it is possible to write:

$$P\left(\bigcup_{l=1}^n A_l\right) = P(A_1) + P\left(\bigcup_{l=2}^n A_l\right) - P\left(\bigcup_{l=2}^n A_1 A_l\right) \quad (\text{A.21})$$

and by (A.19) and (A.20) this corresponds to:

$$\begin{aligned} P\left(\bigcup_{l=1}^n A_l\right) &= P(A_1) + \sum_{i=2}^n P(A_i) - \sum_{2 \leq i < j \leq n} P(A_i A_j) + \dots + \\ &+ \sum_{i=2}^n P(A_1 A_i) - \sum_{2 \leq i < j \leq n} P(A_1 A_i A_j) + \dots = P_1 - P_2 + \dots + (-1)^{n+1} P_n. \end{aligned}$$

The proofs for all  $n$  follows by mathematical induction.

#### A.1.4 Conditional Probability

Given two events  $A$  and  $B$ , it can be interesting to know how the occurrence of one event influences the occurrence of the other one. This relationship is expressed through the *conditional* probability of  $A$  on the hypothesis  $B$ , i.e. the probability of observing  $A$  when  $B$  is known to have occurred:

$$P(A|B) = \frac{P(AB)}{P(B)} \quad (\text{A.22})$$

where  $AB = A \cap B$ . Since  $AB \subseteq B$ , then  $0 \leq P(A|B) \leq 1$ . When  $A$  and  $B$  are mutually exclusive, the intersection  $AB$  is empty and the conditional probability is null. At the other extreme, if  $A \subset B$ , then  $P(A|B) = 1$  because the event  $B$  imply the event  $A$ . If  $A = \bigcup_k A_k$  and the  $A_k$  events are mutually exclusive; then it holds the following *addition law for conditional probabilities*:

$$P(A|B) = \sum_k P(A_k|B). \quad (\text{A.23})$$

It is often convenient to express the probability of an event  $A$  as a sum of conditional probabilities with respect to an *exhaustive set* of mutually exclusive events  $B_k$ , where exhaustive means that  $\bigcup_k B_k = \Omega$ :

$$P(A) = \sum_k P(A|B_k)P(B_k). \quad (\text{A.24})$$

Such equation can be demonstrated by observing that  $A = \bigcup_k AB_k$  and  $P(A)$  can thus be expressed as follows:

$$P(A) = \sum_k P(AB_k) = \sum_k \frac{P(AB_k)}{P(B_k)}P(B_k) \quad (\text{A.25})$$

and, by (A.22), the above expression corresponds to Equation (A.24).

### A.1.5 Statistical Independence

Consider the case of two experiments with different sample spaces  $\Omega_1$  and  $\Omega_2$ . If the experiments are performed always together, it can be interesting to know how the outcome of one experiment is influenced by the outcome of the other one. An example of such a situation is the rolling of two dice; in fact they can be considered as separate experiments leading to separate outcomes. The probability  $P(A_1, A_2)$  of having outcome  $A_1$  for the first experiment and  $A_2$  for the second one can be estimated with the relative frequency:

$$P(A_1, A_2) \simeq \frac{n(A_1, A_2)}{n}. \quad (\text{A.26})$$

If the number of trials  $n$  is sufficiently high and we take into account only the cases where the outcome of the second experiment is  $A_2$ , then we can estimate the probability of observing  $A_1$  as outcome of the first experiment as follows:

$$P(A_1) \simeq \frac{n(A_1, A_2)}{n(A_2)}. \quad (\text{A.27})$$

In fact, as  $n \rightarrow \infty$ ,  $n(A_2)$  tends to the infinity as well and the left side of the above equation corresponds to the relative frequency of the event  $A_1$ . This leads to the following expression for  $P(A_1, A_2)$ :

$$P(A_1, A_2) \simeq \frac{n(A_1, A_2)}{n} = \frac{n(A_1, A_2)}{n(A_2)} \frac{n(A_2)}{n} \simeq P(A_1)P(A_2) \quad (\text{A.28})$$

when two experiments satisfy the above equation when  $n \rightarrow \infty$ , i.e. when  $P(A_1, A_2) = P(A_1)P(A_2)$ , they are said *statistically independent*. On the contrary, when  $P(A_1, A_2) \neq P(A_1)P(A_2)$ , the events are said to be *statistically dependent*.

## A.2 Random Variables

This section provides the main notions about random variables and probability distributions. The rest of this section introduces the concepts of *mean value*, *variance*, *probability distribution* and *covariance*.

### A.2.1 Fundamentals

A variable  $\xi$  is said *random* when its values depend on the events in the sample space of an experiment, i.e. when  $\xi = \xi(\omega)$ . Random variables are associated to functions called *probability distributions* that give, for any couple of values  $x_1$  and  $x_2$  (with  $x_1 \leq x_2$ ), the probability  $P(x_1 \leq \xi \leq x_2)$  of  $\xi$  falling between  $x_1$  and  $x_2$ . When  $\xi$  assumes values belonging to a finite set or to a countable infinity, the variable is called *discrete* and:

$$P(\xi = x) = p_\xi(x) \quad (\text{A.29})$$

where  $p_\xi(x)$  is the probability distribution of  $\xi$ . In this case the probability distribution is discrete as well and:

$$P(x_1 \leq \xi \leq x_2) = \sum_{x=x_1}^{x_2} p_\xi(x) \quad (\text{A.30})$$

where the sum is carried over all values between  $x_1$  and  $x_2$ . If the sum is carried over all possible values of  $\xi$ , i.e. over the whole sample space underlying  $\xi$ , then the result is 1:

$$\sum_{x=-\infty}^{\infty} p_\xi(x) = 1. \quad (\text{A.31})$$

When a random variable takes values in a continuous range, then it is said *continuous* and its distribution function is continuous as well:

$$P(x_1 \leq \xi \leq x_2) = \int_{x_1}^{x_2} p_\xi(x) dx. \quad (\text{A.32})$$

where  $p_\xi(x)$  is called the *probability density function*. If the integration domain covers the whole range of  $x$ , i.e. the whole sample space of the experiment underpinning  $\xi$ , then the result is 1:

$$\int_{-\infty}^{\infty} p_{\xi}(x) dx = 1. \quad (\text{A.33})$$

While in the case of discrete variables it is possible to assign a probability to each value that  $\xi$  can take, in the case of the random variables it is only possible to have the probability  $p_{\xi}(x)dx$  of  $\xi$  falling in a  $dx$  wide interval around  $x$ , i.e. of  $\xi - x$  being smaller than an arbitrary value  $\epsilon$ .

At each probability distribution function corresponds a *cumulative probability function*  $F(x)$  that gives the probability  $P(\xi \leq x)$  of  $\xi$  being less than  $x$ . In the case of discrete variables,  $F(x)$  is a staircase function and it corresponds to the following sum:

$$F(x) = \sum_{x'=-\infty}^x p_{\xi}(x'). \quad (\text{A.34})$$

In the case of continuous random variables,  $F(x)$  is:

$$F(x) = \int_{-\infty}^x p_{\xi}(x') dx' \quad (\text{A.35})$$

and it is a continuous function.

Consider now the *random point*  $\xi = (\xi_1, \xi_2)$ . The probability of  $\xi$  corresponding to a point  $(x_1, x_2)$  is given by the *joint probability distribution*  $p_{\xi_1 \xi_2}(x_1, x_2)$ :

$$p_{\xi_1 \xi_2}(x_1, x_2) = P(\xi_1 = x_1, \xi_2 = x_2). \quad (\text{A.36})$$

The probability  $P(x'_1 \leq \xi_1 \leq x''_1, x'_2 \leq \xi_2 \leq x''_2)$  can be obtained by summing over the corresponding probabilities:

$$P(x'_1 \leq \xi_1 \leq x''_1, x'_2 \leq \xi_2 \leq x''_2) = \sum_{x_1=x'_1}^{x''_1} \sum_{x_2=x'_2}^{x''_2} p_{\xi_1 \xi_2}(x_1, x_2), \quad (\text{A.37})$$

the above is the probability of  $\xi$  falling in the region enclosed by the lines  $\xi_1 = x'_1$ ,  $\xi_1 = x''_1$ ,  $\xi_2 = x'_2$  and  $\xi_2 = x''_2$ . When  $\xi_1$  and  $\xi_2$  are continuous variables, the sums are replaced by integrals and the above probability is written as follows:

$$P(x'_1 \leq \xi_1 \leq x''_1, x'_2 \leq \xi_2 \leq x''_2) = \int_{x'_1}^{x''_1} \int_{x'_2}^{x''_2} p_{\xi_1 \xi_2}(x_1, x_2) dx_1 dx_2 \quad (\text{A.38})$$

where  $p_{\xi_1 \xi_2}(x_1, x_2)$  is called *joint probability density*.

The definitions given for two-dimensional random points can be extended to  $n$ -dimensional points corresponding to  $n$ -tuples of discrete or continuous random variables.

### A.2.2 Mathematical Expectation

The *mathematical expectation* or *mean value*  $\mathcal{E}[\xi]$  of a discrete random variable  $\xi$  corresponds to the following expression:

$$\mathcal{E}[\xi] = \sum_{x=-\infty}^{x=\infty} xp_{\xi}(x) \quad (\text{A.39})$$

where the series is supposed to converge absolutely, i.e. it holds the following:

$$\sum_{x=-\infty}^{x=\infty} |x|p_{\xi}(x) < \infty. \quad (\text{A.40})$$

A variable  $\eta = \phi(x)$ , where  $\phi(x)$  is some function of  $x$ , is a random variable and  $P(\eta = y)$  can be obtained as a sum of the  $p_{\xi}(x)$  over the  $x$  values such that  $\phi(x) = y$ :

$$P(\eta = y) = \sum_{x:\phi(x)=y} p_{\xi}(x) \quad (\text{A.41})$$

The mathematical expectation  $\mathcal{E}[\eta]$  of  $\eta$  can thus be obtained as follows:

$$\mathcal{E}[\eta] = \sum_{y=-\infty}^{y=\infty} yP(\eta = y) = \sum_{y=-\infty}^{y=\infty} y \sum_{x:\phi(x)=y} p_{\xi}(x) = \sum_{x=-\infty}^{x=\infty} \phi(x)p_{\xi}(x) \quad (\text{A.42})$$

and the above definition can be extended to a function of an arbitrary number  $n$  of random variables  $\phi(\xi_1, \xi_2, \dots, \xi_n)$ :

$$\mathcal{E}[\phi(\xi_1, \xi_2, \dots, \xi_n)] = \sum_{x_1=-\infty}^{\infty} \dots \sum_{x_n=-\infty}^{\infty} \phi(x_1, x_2, \dots, x_n)p_{\xi_1\xi_2\dots\xi_n}(x_1, \dots, x_n) \quad (\text{A.43})$$

The mean value of a linear combination of random variables is given by the linear combination of the mean values of the single variables:

$$\mathcal{E}[a\xi_1 + b\xi_2] = a\mathcal{E}[\xi_1] + b\mathcal{E}[\xi_2]. \quad (\text{A.44})$$

In fact, based on Equation (A.43), we can write:

$$\begin{aligned} \mathcal{E}[a\xi_1 + b\xi_2] &= \sum_{x_1=-\infty}^{\infty} \sum_{x_2=-\infty}^{\infty} (ax_1 + bx_2)p_{\xi_1\xi_2}(x_1, x_2) = \\ &= a \sum_{x_1=-\infty}^{\infty} \sum_{x_2=-\infty}^{\infty} x_1p_{\xi_1\xi_2}(x_1, x_2) + b \sum_{x_1=-\infty}^{\infty} \sum_{x_2=-\infty}^{\infty} x_2p_{\xi_1\xi_2}(x_1, x_2) = \\ &= a\mathcal{E}[\xi_1] + b\mathcal{E}[\xi_2]. \end{aligned}$$

When  $\xi_1$  and  $\xi_2$  are independent:

$$\mathcal{E}[\xi_1 \xi_2] = \sum_{x_1=-\infty}^{\infty} \sum_{x_2=-\infty}^{\infty} x_1 x_2 p_{\xi_1}(x_1) p_{\xi_2}(x_2) = \mathcal{E}[\xi_1] \mathcal{E}[\xi_2]. \quad (\text{A.45})$$

When  $\xi$  is continuous, then the mathematical expectation is obtained as an integral:

$$\mathcal{E}[\xi] = \int_{-\infty}^{\infty} x p_{\xi}(x) dx. \quad (\text{A.46})$$

For the variable  $\eta = \phi(\xi)$ , the mathematical expectation is:

$$\mathcal{E}[\eta] = \int_{-\infty}^{\infty} \phi(x) p_{\xi}(x) dx, \quad (\text{A.47})$$

the demonstration follows the same steps as for the corresponding property of discrete variables (see above). The same applies for the mean value of a function  $\phi(\xi_1, \dots, \xi_n)$  of an arbitrary number  $n$  of random variables:

$$\mathcal{E}[\phi(\xi_1, \dots, \xi_n)] = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} \phi(\xi_1, \dots, \xi_n) p_{\xi_1 \dots \xi_n}(x_1, \dots, x_n) dx_1 \dots dx_n. \quad (\text{A.48})$$

The properties demonstrated for the discrete variables can be demonstrated also for the continuous ones by replacing sums with integrals. This is possible because the formal properties of sums and integrals are the same.

### A.2.3 Variance and Covariance

The *variance* (or *dispersion*)  $D[\xi]$  of a random variable is the mathematical expectation  $\mathcal{E}[(\xi - \mu)^2]$  of the quantity  $(\xi - \mu)^2$ , where  $\mu = \mathcal{E}[\xi]$ . The variance expression for a discrete variable is

$$D[\xi] = \mathcal{E}[(\xi - \mu)^2] = \sum_{x=-\infty}^{\infty} (x - \mu)^2 p_{\xi}(x) \quad (\text{A.49})$$

while for a continuous variable it is:

$$D[\xi] = \mathcal{E}[(\xi - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 p_{\xi}(x) dx. \quad (\text{A.50})$$

The properties of the variance can be demonstrated without distinguishing between continuous and discrete random variables; in fact they are mostly based on the properties of the mathematical expectation that have the same form for both continuous and discrete variables. It follows from the definition that:

$$D[\xi] = \mathcal{E}[(\xi - \mu)^2] = \mathcal{E}[(\xi^2 - 2\mu\xi + \mu^2)] = \mathcal{E}[\xi^2] - 2\mu\mathcal{E}[\xi] + \mu^2 = \mathcal{E}[\xi^2] - \mu^2, \quad (\text{A.51})$$

then



$$D[c\xi] = \mathcal{E}[c^2\xi^2] - (\mathcal{E}[c\xi])^2 = c^2D[\xi] \quad (\text{A.52})$$

because  $\mathcal{E}[c\xi] = c\mathcal{E}[\xi]$  (see the previous section).

If  $\xi_1$  and  $\xi_2$  are two independent random variables, then:

$$\begin{aligned} D[\xi_1 + \xi_2] &= \mathcal{E}[(\xi_1 + \xi_2 - \mu_1 - \mu_2)^2] = \\ &= \mathcal{E}[(\xi_1 - \mu_1)^2] + \mathcal{E}[(\xi_2 - \mu_2)^2] + 2\mathcal{E}[(\xi_1 - \mu_1)(\xi_2 - \mu_2)], \\ &= D[\xi_1] + D[\xi_2] + 2\mathcal{E}[(\xi_1 - \mu_1)]\mathcal{E}[(\xi_2 - \mu_2)], \end{aligned}$$

since  $\mathcal{E}[(\xi_i - \mu_i)] = 0$ , the above corresponds to:

$$D[\xi_1 + \xi_2] = D[\xi_1] + D[\xi_2]. \quad (\text{A.53})$$

Consider a random point  $\xi = (\xi_1, \xi_2, \dots, \xi_n)$ , the mathematical expectation of the product  $(\xi_i - \mu_i)(\xi_j - \mu_j)$ , where  $\mu_i$  and  $\mu_j$  are the mean values of  $\xi_i$  and  $\xi_j$ , respectively, is called *covariance*  $\sigma_{ij}$  of  $\xi_i$  and  $\xi_j$ :

$$\sigma_{ij} = \mathcal{E}[(\xi_i - \mu_i)(\xi_j - \mu_j)], \quad (\text{A.54})$$

based on the above definition,  $\sigma_{ii} = D[\xi_i]$ . The  $n \times n$  matrix  $\Sigma$  such that  $\Sigma_{ij} = \sigma_{ij}$  is called *covariance matrix* of  $\xi$  and it has the variances of the  $\xi_i$  variables on the main diagonal.



## B

---

# Signal Processing

## B.1 Introduction

The goal of this appendix is to provide basic notions about signal processing, the domain involving mathematical techniques capable of extracting from signals information useful for several tasks. The data considered in this book, i.e. audio recordings, images and videos, can be considered as signals and the techniques presented in this appendix are often applied to analyze them. Section B.2 is dedicated to a quick recall of complex numbers because most signal processing techniques include functions defined on the complex domain. Section B.3 is dedicated to the  $z$ -transform, a mathematical approach to represent signals through infinite series of powers that make easier to study the effect of systems (see Section 2.5). Section B.3.2 introduces the *Fourier transform*, a special case of the  $z$ -transform that enables us to analyze the frequency properties of signals. Section B.3.3 presents the Discrete Fourier Transform, a representation for periodic digital signals that can be applied also for finite length generic signals and represents sequences through sums of elementary sines and cosines. Section B.4 describes the *discrete cosine transform*, a representation commonly applied in image processing and close to the Discrete Fourier Transform.

The content of this appendix is particularly useful for understanding Chapter 2, Chapter 3 and Chapter 12.

## B.2 The Complex Numbers

The *complex numbers* are an extension of the real numbers containing all roots of quadratic equations. If  $j$  is the solution of the following equation:

$$x^2 = -1 \tag{B.1}$$

then the set  $\mathbf{C}$  of complex numbers is represented in *standard form* as:

$$\{a + bj : a, b \in \mathbf{R}\} \quad (\text{B.2})$$

where the symbol  $:$  stands for *such that* and  $\mathbf{R}$  is the set of the real numbers. A complex number is typically expressed with a single variable  $z$ , the number  $a$  is called *real part*  $Re(z)$  of  $z$ , and  $b$  is called the *imaginary part*  $Im(z)$  of  $z$ . The plan having as coordinates the values of  $a$  and  $b$  is called *complex* or  $z$  plan. Each point of such plan is a complex number and, vice versa, all complex numbers correspond to one point of such plan. The horizontal axis of the  $z$  plan is called the *real axis*, while the vertical one is defined *imaginary axis*. The sum and product between complex numbers are defined as follows:

$$(a + bj) + (c + dj) = (a + b) + (c + d)j \quad (\text{B.3})$$

$$(a + bj)(c + dj) = (ac - bd) + (ad + bc)j \quad (\text{B.4})$$

where the fact that  $j^2 = -1$  is applied. Two complex numbers  $z_1$  and  $z_2$  that have the same real part  $a$ , but imaginary parts  $b$  and  $-b$ , respectively, are said to be *complex conjugates* and this is expressed by writing  $z_2 = z_1^*$ .

Since the complex numbers can be interpreted as vectors in the  $z$  plan, it is possible to define their *modulus*<sup>1</sup>  $|z|$  as follows:

$$|z| = \sqrt{a^2 + b^2}. \quad (\text{B.5})$$

The modulus can be calculated as  $|z| = \sqrt{zz^*}$  and, as a consequence,  $|z| = |z^*|$ .

Since the complex numbers can be thought of as vectors in the  $z$  plan, it is possible to express them in *polar form* (see Figure B.1). In fact, if  $r = |z|$  and  $\tan \theta = b/a$ , then  $a = r \cos \theta$  and  $b = r \sin \theta$ , and by the Euler's equation:

$$e^{j\theta} = r \cos \theta + j \sin \theta, \quad (\text{B.6})$$

it is possible to write:

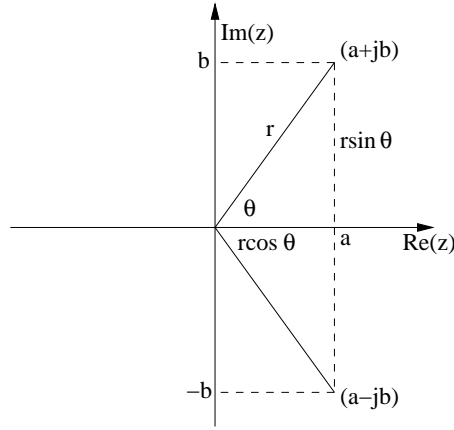
$$z = re^{j\theta}. \quad (\text{B.7})$$

The number  $r$  is called the *magnitude* and the angle  $\theta$  is called *argument* and expressed by  $Arg(z)$ . The argument of a complex number is not unique because  $z$  is not changed by adding integer multiples of  $2\pi$  to  $\theta$ . The argument in the interval  $] -\pi, \pi]$  (where the  $]$  on the left side means that the left extreme is not included) is called *principal value*. The complex conjugate of  $z = r(\cos \theta + j \sin \theta)$  is  $r(\cos \theta - j \sin \theta)$ , i.e.  $z^*$  is obtained by changing  $\theta$  into  $-\theta$ . In other words, two complex conjugates have the same magnitude by opposite arguments.

The complex numbers  $e^{j\theta}$ , with  $\theta \in ] -\pi, \pi]$ , define the so-called *unit circle* in the  $z$  plan. The equation  $z^N = 1$  has  $N$  complex roots with magnitude 1. Since the roots are complex numbers, they can be identified as follows:

---

<sup>1</sup> The modulus is the distance of the point representing a complex number from the origin of the plane where  $a$  and  $b$  are the axes.



**Fig. B.1.** Complex plan. The figure shows a complex number and its complex conjugate in the complex plan. The real and imaginary parts can be expressed in terms of  $r$  and  $\theta$  to obtain the polar form.

$$(e^{j\theta})^N = e^{jN\theta} = 1. \quad (\text{B.8})$$

Since  $e^{j\theta} = 1$  when  $\theta = 2k\pi$  (where  $k$  is an integer), the last equation corresponds to  $N\theta = 2k\pi$ , then:

$$\theta = \frac{2k\pi}{N} \quad (\text{B.9})$$

and the  $N$  roots of 1 are the complex exponentials  $e^{j\frac{2k\pi}{N}}$ , where  $k = 0, 1, \dots, N-1$ .

When  $k > N-1$ , the value of the argument is simply increased by multiples of  $2\pi$  and the roots are the same as those corresponding to the  $k$  values between 0 and  $N-1$ . The roots of 1 are used to represent periodic signals with the discrete Fourier transform (see Section B.3.3).

### B.3 The $z$ -Transform

Given a continuous signal  $s(t)$ , it is possible to obtain, through an A/D conversion including sampling and quantization, a *digital signal*  $\{s[0], \dots, s[N-1]\}$  such that:

$$s[n] = s(nT) = s(n/F) \quad (\text{B.10})$$

where  $n$  is an integer,  $T$  is called sampling period and  $F$  is the sampling frequency. Issues related to sampling (see Section 2.3.1) and quantization (see Section 2.7) have been discussed in Chapter 2. A digital signal of length  $N$ , i.e. including  $N$  samples in the sequence  $\{s[n]\}$ , can be thought of as an infinite length signal such that  $s[n] = 0$  for  $n < 0$  and  $n \geq N$ . In the rest of this

appendix, digital signals will be referred to as signals and denoted with  $s[n]$  whenever there is no ambiguity between sequences and single samples.

The  $z$ -transform of a digital signal  $\{s[n]\}$  is defined by the following pair of equations:

$$S(z) = \sum_{n=-\infty}^{\infty} s[n]z^{-n} \quad (\text{B.11})$$

$$s[n] = \frac{1}{2\pi j} \oint_C S(z)z^{n-1} dz \quad (\text{B.12})$$

where Equation (B.11) defines the *direct* transform, Equation (B.12) defines the *inverse* one and  $C$  is a closed contour that encircles the  $z$  plan origin and lies in the region of existence of  $S(z)$  (see below).

The  $z$ -transform can be seen as an infinite series of powers of the variable  $z^{-1}$  where the  $s[n]$  are the coefficients. The series converges to a finite value when the following sufficient condition is met:

$$\sum_{n=-\infty}^{\infty} |s[n]| |z^{-n}| < \infty. \quad (\text{B.13})$$

The above equation corresponds to a region of the  $z$  plan, called the *region of convergence*, which has the following form:

$$R_1 < |z| < R_2 \quad (\text{B.14})$$

and the values of  $R_1$  and  $R_2$  depend on the characteristics of the sequence  $\{s[n]\}$ . Consider, for example, a rectangular window  $w[n]$  of length  $N$  (see Section 2.5), the  $z$ -transform is:

$$W(z) = \sum_{n=0}^{N-1} z^{-n} = \frac{1 - z^{-N}}{1 - z^{-1}} \quad (\text{B.15})$$

and the region of convergence is  $0 < |z| < \infty$ . Such a result applies to any finite length sequence.

Consider now the sequence  $s[n] = a^n u[n]$ , where  $u[n]$  is 1 for  $n \geq 0$  and 0 otherwise. In this case, the  $z$ -transform is:

$$S(z) = \sum_{n=0}^{\infty} a^n z^{-n} = \frac{1}{1 - az^{-1}} \quad (\text{B.16})$$

and the series converges for  $|z| > |a|$ . This result applies to infinite length sequences which are non-zero only for  $n \geq 0$  and it corresponds to a region of convergence of the form  $|R_1| < |z| < \infty$ .

The case of a sequence different from zero only when  $n < 0$  can be studied by considering the case of  $s[n] = b^n u[-n - 1]$  (where  $u[n]$  is the same function as in the previous example):

$$S[n] = \sum_{n=-\infty}^{-1} b^n z^{-n} = \frac{1}{1 - bz^{-1}}. \quad (\text{B.17})$$

Such series converges for  $|z| < |b|$  and, in terms of Equation B.14, this corresponds to the form  $0 < |z| < R_2$ .

The last example concerns an infinite length sequence which is different from zero for  $-\infty < n < \infty$ . Such case is a combination of the last two examples and it leads to a region of convergence of the form  $R_1 < |z| < R_2$ .

### B.3.1 z-Transform Properties

The z-transform has several properties that are demonstrated in the following. The first is the linearity:

**Theorem B.1.** *If  $s[n] = as_1[n] + bs_2[n]$ , then:*

$$S(z) = aS_1(z) + bS_2(z). \quad (\text{B.18})$$

The demonstration follows directly from the definition of the z-transform:

$$S(z) = \sum_{n=-\infty}^{\infty} (s_1[n] + bs_2[n])z^{-n} = aS_1(z) + bS_2(z) \quad (\text{B.19})$$

Consider the signal  $s[n - n_0]$ , where  $n_0$  is a constant integer. The effect on the z-transform is described by the following theorem.

**Theorem B.2.** *The z-transform  $S_{n_0}(z)$  of a signal  $s_{n_0}[n] = s[n - n_0]$  is related to the z-transform  $S(z)$  of  $s[n]$  through the following relationship:*

$$S_{n_0}(z) = z^{-n_0} S(z). \quad (\text{B.20})$$

The z-transform of  $s_{n_0}[n]$  can be written as:

$$S_{n_0}(z) = \sum_{n=-\infty}^{\infty} s[n - n_0]z^{-n}, \quad (\text{B.21})$$

if  $m = n - n_0$ , then the last equation becomes:

$$S_{n_0}(z) = \sum_{m=-\infty}^{\infty} s[m]z^{-m-n_0} = z^{-n_0} S(z) \quad (\text{B.22})$$

The elements of a sequence can be weighted with an exponential resulting into a signal  $s_a[n] = a^n s[n]$ . The effect on the z-transform is as follows:

**Theorem B.3.** *The z-transform  $S_a(z)$  of the signal  $s_a[n] = a^n s[n]$  is related to the z-transform  $S(z)$  of  $s[n]$  through the following relationship:*

$$S_a(z) = S(za^{-1}). \quad (\text{B.23})$$

Following the definition of the  $z$ -transform, it is possible to write that:

$$S_a(z) = \sum_{n=-\infty}^{\infty} a^n s[n] z^{-n} = \sum_{n=-\infty}^{\infty} s[n] \left(\frac{z}{a}\right)^{-n} = S(za^{-1}) \quad (\text{B.24})$$

**Theorem B.4.** *The  $z$ -transform  $S_n(z)$  of the signal  $ns[n]$  is related to the  $z$ -transform  $S(z)$  of  $s[n]$  through the following expression:*

$$S_n(z) = -z \frac{dS(z)}{dz}. \quad (\text{B.25})$$

The expression of  $S_n(z)$  is:

$$S_n(z) = \sum_{n=-\infty}^{\infty} ns[n] z^{-n} = z \sum_{n=-\infty}^{\infty} ns[n] z^{-n-1}. \quad (\text{B.26})$$

Since  $-nz^{-n-1}$  is the derivative of  $z^{-n}$ , the above corresponds to:

$$S_n(z) = -z \sum_{n=-\infty}^{\infty} s[n] \frac{d(z^{-n})}{dz} = -z \frac{dS(z)}{dz} \quad (\text{B.27})$$

**Theorem B.5.** *The  $z$ -transform  $S_-(z)$  of the signal  $s[-n]$  is related to the  $z$ -transform  $S(z)$  of the signal  $s[n]$  through the following expression:*

$$S_-(z) = S(z^{-1}). \quad (\text{B.28})$$

Following the definition of the  $z$ -transform:

$$S_-(z) = \sum_{n=-\infty}^{\infty} s[-n] z^{-n}. \quad (\text{B.29})$$

If  $m = -n$ , the above equation becomes:

$$S_-(z) = \sum_{m=-\infty}^{\infty} s[m] z^m = \sum_{m=-\infty}^{\infty} s[m] \left(\frac{1}{z}\right)^{-m} = S(z^{-1}) \quad (\text{B.30})$$

**Theorem B.6.** *The  $z$ -transform  $C(z)$  of the convolution of two digital signals  $c[n] = s[n] * h[n]$  corresponds to the product of the  $z$ -transforms  $S(z)$  and  $H(z)$  of  $s[n]$  and  $h[n]$ , respectively:*

$$C(z) = S(z)H(z). \quad (\text{B.31})$$

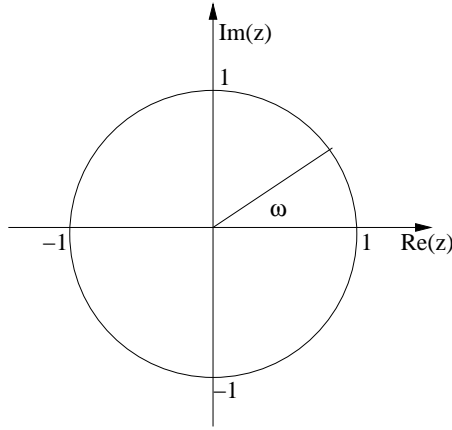
The convolution between  $s[n]$  and  $h[n]$  is  $c[n] = \sum_{k=-\infty}^{\infty} s[k]h[n-k]$ , thus the  $z$ -transform of  $c[n]$  is:

$$C(z) = \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} z^{-n} s[k]h[n-k] = \sum_{k=-\infty}^{\infty} s[k] \sum_{n=-\infty}^{\infty} h[n-k] z^{-n}. \quad (\text{B.32})$$

If  $n-k = m$ , the above expression can be rewritten as:

$$C(z) = \sum_{k=-\infty}^{\infty} s[k] z^{-k} \sum_{m=-\infty}^{\infty} h[m] z^{-m} = S(z)H(z) \quad (\text{B.33})$$





**Fig. B.2.** Unit circle. The figure shows the unit circle in the  $z$  plan. The angle  $\omega$  identifies a point on the unit circle.

### B.3.2 The Fourier Transform

The *Fourier Transform* (FT) is defined through the following two equations:

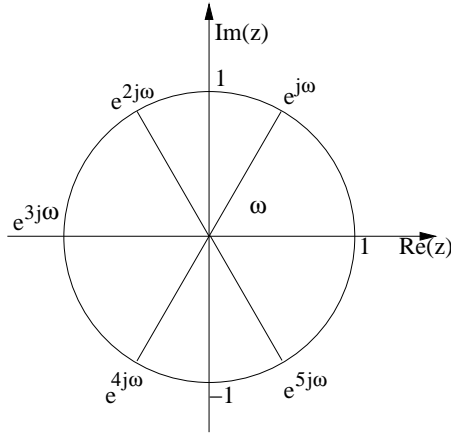
$$S(e^{j\omega}) = \sum_{n=-\infty}^{\infty} s[n]e^{-j\omega n} \quad (\text{B.34})$$

$$s[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} S(e^{j\omega})e^{j\omega n} d\omega \quad (\text{B.35})$$

where Equation (B.34) defines the inverse transform and Equation (B.35) defines the inverse one. The FT corresponds to the  $z$ -transform when  $z = e^{j\omega}$ , i.e. when  $z$  lies on the unit circle of the  $z$  plan. Since  $|e^{j\omega}| = 1$ , the condition for the existence of the FT is (see Equation (B.13)):

$$\sum_{n=-\infty}^{\infty} |s[n]| < \infty. \quad (\text{B.36})$$

The region of convergence of the above series can be deduced from the examples described in Section B.3 by posing  $z = e^{j\omega}$ . This corresponds to impose as a condition that the unit circle lies in the region of convergence  $R_1 < |z| < R_2$ . In the case of finite-length sequences, when  $R_1 = 0$  and  $R_2 = \infty$ , the FT exists always. For sequences different from zero only when  $n \geq 0$ , the region of convergence is  $R_1 < |z| < \infty$  and the FT exists when  $R_1 < 1$ . For infinite-length sequences different from zero when  $n < 0$ ,  $R_1 = 0$  and  $R_2$  is a finite constant; thus the FT exists when  $R_2 > 1$ . For the last example in Section B.3, i.e. an infinite length sequence different from zero for both  $n < 0$  and  $n \geq 0$ , both  $R_1$  and  $R_2$  are finite constants and the FT exists when  $R_1 < 1 < R_2$ .



**Fig. B.3.** DFT interpretation. The DFT can be thought of as a sampling of the  $z$ -transform along the unit circle. The figure shows the points corresponding to integer multiples of the angle  $\omega = \pi/6$ , where the  $z$ -transform is sampled in the case of period 6.

An important aspect of the FT is that it is a periodic function of  $\omega$  with period  $2\pi$ . This can be shown by replacing  $\omega$  with  $\omega + 2\pi$  in Equations (B.11) and (B.12), but also by observing that  $\omega$  determines the position on the unit circle of the  $z$  plan (see Figure B.2). When  $\omega$  is increased by an integer multiple of  $2\pi$ , the position on the unit circle is always the same; thus the FT has the same value.

The properties demonstrated in Section B.3.1 for the  $z$ -transform can be extended to the FT by simply replacing  $z$  with  $e^{j\omega}$ . However, the properties hold only when the FTs exist.

### B.3.3 The Discrete Fourier Transform

If a digital signal  $\hat{s}[n]$  is periodic with period  $N$ , i.e.  $\hat{s}[n] = \hat{s}[n + N]$  for  $-\infty < n < \infty$ , then it can be represented by a Fourier series:

$$\hat{S}[k] = \sum_{n=0}^{N-1} \hat{s}[n] e^{-j\frac{2\pi}{N}kn} \quad (\text{B.37})$$

$$\hat{s}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{S}[k] e^{j\frac{2\pi}{N}kn} \quad (\text{B.38})$$

where Equation (B.37) defines the direct transform and Equation (B.38) defines the inverse one. The *discrete Fourier transform* (DFT) is an exact representation for any periodic digital signal, but it can be used, with some precautions, to represent finite-length nonperiodic sequences. In fact, consider

the  $z$ -transform of a digital signal  $s[n]$  which is equal to zero for  $n < 0$  and  $n \geq N$ :

$$S(z) = \sum_{n=0}^{N-1} s[n]z^{-n}, \quad (\text{B.39})$$

if  $z = e^{j\frac{2\pi}{N}k}$ , the above equation becomes:

$$S(e^{j\frac{2\pi}{N}k}) = \sum_{n=0}^{N-1} s[n]e^{-j\frac{2\pi}{N}kn}, \quad (\text{B.40})$$

i.e. it corresponds to the  $\hat{S}[k]$  value for a periodic signal  $\hat{s}[n]$  obtained by replicating infinite times  $s[n]$ . In other words, given a finite length signal  $s[n]$ , it is possible to create an infinite length periodic signal  $\hat{s}[n]$  such that  $\hat{s}[n+rN] = s[n]$ , where  $r$  is an integer. The DFT is an exact representation of  $\hat{s}[n]$ , but it can be used to represent  $s[n]$  when only the intervals  $0 \leq n \leq N-1$  and  $0 \leq k \leq N-1$  are taken into account. Equation (B.40) can be thought of as a sampling of the  $z$ -transform on the unit circle of the  $z$  plan (see Figure B.3). For this reason, the properties of the DFT are the same as those of the  $z$ -transform with the constraint that  $z = \exp(-2j\pi kn/N)$ .

## B.4 The Discrete Cosine Transform

The discrete cosine transform (DCT) is commonly applied in image coding and can be computed via the DFT. Given the  $N$  long signal  $s[n]$ ,  $0 \leq n < N$ , it is possible to obtain a signal  $s_e[n]$  of length  $2N$  in the following way:

$$s_e[n] = \begin{cases} s[n] & 0 \leq n < N \\ 0 & N \leq n < 2N - 1. \end{cases} \quad (\text{B.41})$$

The signal  $s_e[n]$  can then be used to create a  $2N$  long sequence  $y[n]$  defined as:

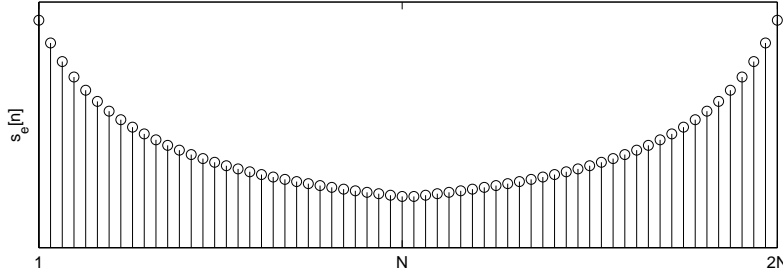
$$y[n] = s_e[n] + s_e[2N - 1 - n], \quad (\text{B.42})$$

i.e. a symmetric signal where the first  $N$  samples correspond to those of the original  $s[n]$  sequence and the remaining  $N$  correspond to the same samples, but in a reversed order (see Figure B.4).

The DFT of  $y[n]$  can be written as follows:

$$Y[k] = \sum_{n=0}^{2N-1} y[n]e^{-j\frac{2\pi}{2N}kn}, \quad (\text{B.43})$$

but by definition (see Equation (B.42)),  $y[n] = y[2N - 1 - n]$ , then the DFT of  $y[n]$  can be rewritten as:



**Fig. B.4.** Extended signal. The plot shows a signal obtained by adding  $s_e[n]$  and  $s_e[2N - 1 - n]$ .

$$Y[k] = \sum_{n=0}^{N-1} s[n](e^{-j\frac{2\pi}{2N}kn} + e^{-j\frac{2\pi}{2N}k(2N-n-1)}). \quad (\text{B.44})$$

The  $N$  point DCT  $C[k]$  of  $s[n]$  is then defined as:

$$C(k) = \begin{cases} Y[k]e^{-j\frac{\pi}{2N}kn} & 0 \leq k < N. \\ 0 & \text{otherwise.} \end{cases} \quad (\text{B.45})$$

By plugging Equation (B.43) into the definition of  $C(k)$ , the result is (for  $0 \leq k < N$ ):

$$C(k) = \sum_{n=0}^{N-1} 2s[n] \cos\left(\frac{(2n+1)k\pi}{2N}\right). \quad (\text{B.46})$$

One of the most important aspects of the DCT is that its coefficients are always real, while in the case of the DFT they are typically complex. The DCT defined in this section is often referred to as *even symmetrical* DCT.

The inverse transform requires as a first step the definition of a  $2N$ -point DFT  $Y[k]$ :

$$Y(k) = \begin{cases} C[k]e^{-j\frac{2\pi}{2N}\frac{k}{2}} & 0 \leq k < N \\ 0 & k = N \\ -C[2N - k]e^{-j\frac{2\pi}{2N}\frac{k}{2}} & N + 1 \leq k \leq 2N - 1. \end{cases} \quad (\text{B.47})$$

This enables us to obtain the inverse DFT as follows:

$$y[n] = \frac{1}{2N} \sum_{k=0}^{2N-1} Y[k]e^{j\frac{2\pi}{2N}kn} \quad (\text{B.48})$$

where  $0 \leq n \leq 2N - 1$ , and the inverse DCT corresponds to the following:

$$s[n] = \begin{cases} y[n] & 0 \leq n \leq N - 1 \\ 0 & \text{otherwise.} \end{cases} \quad (\text{B.49})$$

By definition (see (Equation B.47))  $C[k] = C[2N - k]$  ( $k = 1, \dots, N - 1$ ), then  $y[n]$  can be rewritten as a sum over  $N$  elements:

$$y[n] = \frac{1}{2N}C[0] + \frac{1}{2N} \sum_{k=1}^{N-1} C[k] (e^{-j\frac{\pi k}{2N}(2n+1)} - e^{-j\frac{\pi}{2N}(2N-k)(2n+1)}) = \quad (\text{B.50})$$

$$= \frac{1}{2N}C[0] + \frac{1}{N} \sum_{k=1}^{N-1} C[k] \cos\left(\frac{\pi k(2n+1)}{2N}\right). \quad (\text{B.51})$$

In other words, we can write the inverse DCT as:

$$s[n] \begin{cases} \frac{1}{N} \sum_{k=1}^{N-1} \alpha(k) C[k] \cos\left(\frac{\pi k(2n+1)}{2N}\right) & 0 \leq n \leq N - 1 \\ 0 & \text{otherwise,} \end{cases} \quad (\text{B.52})$$

where  $\alpha(k) = 1/2$  for  $k = 0$  and  $\alpha(k) = 1$  for  $k = 1$ .



# C

---

## Matrix Algebra

### C.1 Introduction

The goal of this appendix is to provide the main notions about matrix algebra and eigenvector calculation. Section C.2 introduces basic definitions and matrix operations, Section C.3 shows matrix determinants and their properties and Section C.4 presents eigenvalues and eigenvectors.

### C.2 Fundamentals

An  $m \times n$  matrix is a rectangular array of numbers composed of  $m$  rows and  $n$  columns:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad (\text{C.1})$$

where the  $a_{ij}$  entries are called *elements*. The above expression is often written in the following more compact form:

$$A = [a_{ij}]. \quad (\text{C.2})$$

When  $m = n$ , i.e. the number of columns and rows is the same, the matrix is said to be *square* and the elements  $a_{ii}$ , where  $i = 1, 2, \dots, n$ , form the *main diagonal* of the matrix. The following sum:

$$T(A) = \sum_{k=1}^n a_{kk} \quad (\text{C.3})$$

is called the *trace* of the matrix. The trace can be calculated only for square matrices because only these have a main diagonal. Given a matrix  $A = [a_{ij}]$ , the matrix  $A^T = [a_{ji}]$  is called the *transpose* of  $A$  and it can be obtained by

interchanging rows and columns of  $A$ . If  $A^T = A$ , the matrix is said *symmetric*. The transpose of an  $m \times n$  matrix is an  $n \times m$  one, thus a matrix cannot be symmetric if  $m \neq n$ , i.e. if it is not square.

Given two matrices  $A = [a_{ij}]$  and  $B = [b_{ij}]$ , their sum is defined as follows:

$$A + B = [a_{ij} + b_{ij}]. \quad (\text{C.4})$$

In other words, the element  $ij$  of the sum corresponds to the sum of the  $ij$  elements of  $A$  and  $B$ . The subtraction  $A - B$  corresponds to the matrix where the element  $ij$  is  $a_{ij} - b_{ij}$ :

$$A - B = [a_{ij} - b_{ij}]. \quad (\text{C.5})$$

The multiplication of a matrix  $A$  by a scalar  $c$  is defined as follows:

$$cA = [ca_{ij}], \quad (\text{C.6})$$

the result of such an operation is that each element of  $A$  is multiplied by  $c$ . The multiplication  $A \cdot B$  between two matrices is calculated as follows:

$$A \cdot B = \left[ \sum_k a_{ik} b_{kj} \right], \quad (\text{C.7})$$

i.e. the element  $ij$  corresponds to the dot product of the  $i$ th row of  $A$  and of the  $j$ th column of  $B$ . This means that two matrices can be multiplied only when the number of columns of the first one is equal to the number of rows of the second one. A matrix  $\mathbf{I}$  such that  $A \cdot \mathbf{I} = A$  is called *identity matrix*.

Given two matrices  $X$  and  $Y$ , if the following holds:

$$XY = YX = \mathbf{I} \quad (\text{C.8})$$

then  $Y$  is the *inverse matrix* of  $X$  and viceversa. Only square matrices can be inverted because this is the only case where both  $XY$  and  $YX$  have the same number of rows and columns. In the case of a rectangular matrix  $X$ , it is possible to define the so-called *Moore Penrose pseudoinverse*  $\hat{X}$ :

$$\hat{X} = X^T (X^T X)^{-1}. \quad (\text{C.9})$$

### C.3 Determinants

Consider the following  $2 \times 2$  square matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ c_{21} & a_{22} \end{pmatrix} \quad (\text{C.10})$$

the expression  $\det(A) = a_{11}a_{22} - a_{12}a_{21}$  is called *determinant* of  $A$ . If the matrix is  $3 \times 3$ , then the determinant can be obtained as follows:



$$\det(A) = \sum_{i=1}^3 (-1)^{i+j} a_{ij} \det(A_{ij}) = \sum_{j=1}^3 (-1)^{i+j} a_{ij} \det(A_{ij}) \quad (\text{C.11})$$

where  $i$  and  $j$  can be selected arbitrarily and  $A_{ij}$  is a matrix obtained by removing column  $j$  and row  $i$  from  $A$ . In other words, the calculation of the determinant is performed by first selecting one column (or one row) arbitrarily, and then by multiplying its elements  $ij$  by  $(-1)^{i+j} \det(A_{ij})$ . In order to simplify the calculations, it is advised to select the row or the column with the highest number of null elements. In fact this minimizes the number of addends of the sums in Equation (C.11). When  $A$  is  $n \times n$  with  $n > 3$ ,  $\det(A)$  can be obtained recursively starting from the above expressions. Only square matrices have a determinant because only in this case the iterative removal of one column and one row brings to a  $2 \times 2$  matrix.

The above technique is formalized in the *Laplace expansion theorem*:

**Theorem C.1.** *The determinant of a matrix  $A$  can be calculated as follows:*

$$\det(A) = \sum M_k A_{n-k} \quad (\text{C.12})$$

where the sum goes over all determinants  $M_k$  of order  $k$  that can be formed of rows  $i_1, \dots, i_k$  and columns  $j_1, \dots, j_k$ , and  $A_{n-k}$  is the product of the number  $(-1)^{i_1+\dots+i_k+j_1+\dots+j_k}$  and the determinant of the matrix remaining from  $A$  by deleting the rows  $i_1, \dots, i_k$  and the columns  $j_1, \dots, j_k$  used to form  $M_k$ .

The demonstration is omitted and the reader can refer to any academic textbook on matrix algebra.

The determinant of a matrix has several properties that are shown and proved in the following.

**Theorem C.2.** *If  $A$  is an  $n \times n$  matrix and  $c$  is a scalar, then  $\det(cA) = c^n \det(A)$ .*

The proof can be obtained inductively. In the case of the  $2 \times 2$  matrix of Equation (C.10), the expression of  $\det(cA)$  is as follows:

$$\det(cA) = ca_{11}ca_{22} - ca_{12}ca_{21} = c^2(a_{11}a_{22} - a_{12}a_{21}) = c^2 \det(A). \quad (\text{C.13})$$

If the property holds for an  $n-1 \times n-1$  matrix, then the determinant of  $cA$ , where  $A$  is an  $n \times n$  matrix, can be calculated as follows:

$$\det(cA) = \sum_{i=1}^n (-1)^{i+j} ca_{ij} \det(cA_{ij}) = \sum_{i=1}^n (-1)^{i+j} ca_{ij} c^{n-1} \det(A_{ij}) = c^n \det(A), \quad (\text{C.14})$$

and this demonstrates the theorem.

**Theorem C.3.** *If  $A$  and  $B$  are  $n \times n$  square matrices, then  $\det(AB) = \det(A)\det(B)$ .*

**Theorem C.4.** *If  $A^T$  is the transpose of  $A$ , then  $\det(A^T) = \det(A)$ .*

The demonstration can be obtained by induction. When  $A$  is a  $2 \times 2$  matrix like in Equation (C.10), then  $A^T$  is as follows:

$$A = \begin{pmatrix} a_{11} & a_{21} \\ c_{12} & a_{22} \end{pmatrix}. \quad (\text{C.15})$$

By the definition of determinant:

$$\det(A) = a_{11}a_{22} - a_{12}a_{21} = \det(A^T) \quad (\text{C.16})$$

and the theorem is demonstrated for  $n = 2$ . If  $A$  is now an  $n + 1 \times n + 1$  matrix, its determinant can be obtained as:

$$\det(A) = \sum_{i=1}^{n+1} (-1)^{i+j} a_{ij} \det(A_{ij}) = \sum_{i=1}^{n+1} (-1)^{i+j} (A^T)_{ji} \det(A_{ji}^T) = \det(A^T) \quad (\text{C.17})$$

where the last passage is based on the fact that  $(A_{ij})^T = A_{ji}^T$ .

**Theorem C.5.** *Consider an  $n \times n$  matrix  $A$ ,  $\det(A) \neq 0$  if and only if  $A$  is nonsingular.*

The first step is to demonstrate that if  $A$  is nonsingular, then  $\det(A) \neq 0$ . If  $A$  is non singular,  $A^{-1}$  exists and  $AA^{-1} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix. By property C.3:

$$\det(AA^{-1}) = \det(A)\det(A^{-1}) = \det(\mathbf{I}) = 1 \quad (\text{C.18})$$

and this is possible only if  $\det(A) \neq 0$ .

The second step is to prove that if  $\det(A) \neq 0$ , then  $A^{-1}$  exists. The demonstration can be made by contradiction. If  $\det(A) = 0$  and  $A^{-1}$  exists, then  $\det(A)\det(A^{-1}) = 1$  (see above), but this is not possible because  $\det(A) = 0$ .

## C.4 Eigenvalues and Eigenvectors

Consider the square matrix  $A$ , the scalar  $\lambda$  is defined *eigenvector* of  $A$  if it exists a nonzero vector  $x$  (called *eigenvector*) such that:

$$Ax = \lambda x, \quad (\text{C.19})$$

where  $\lambda$  and  $x$  are said to form an *eigenpair*. If  $x$  is an eigenvector of  $A$ , then any other vector  $cx$ , where  $c$  is a scalar, is an eigenvector of  $A$ :

$$A(cx) = cAx = c\lambda x = \lambda(cx). \quad (\text{C.20})$$

The eigenvectors form the basis of a vectorial space called *eigenspace*. When  $\lambda = 0$ , the eigenspace is called *null space* or *kernel* of  $A$ . When  $A$  is the identity

matrix  $\mathbf{I}$ , the equation  $\mathbf{I}x = x$  is always satisfied, i.e. all  $n$ -dimensional vectors are eigenvectors (with  $\lambda = 1$ ) of the  $n \times n$  identity matrix.

The eigenpairs can be found by solving the equation  $Ax = \lambda x$  that can be rewritten as follows:

$$(A - \lambda\mathbf{I})x = 0. \quad (\text{C.21})$$

where the second member is the null vector. The eigenvectors form the null space of the matrix  $A - \lambda\mathbf{I}$  and they can be known once the eigenvalues are available. On the other hand, the above equation can have nonzero solutions only if  $A - \lambda\mathbf{I}$  is singular, i.e. if

$$\det(A - \lambda\mathbf{I}) = 0. \quad (\text{C.22})$$

The above *characteristic equation* involves  $\lambda$ , but not  $x$ ; however, the eigenvectors can be obtained when the last equation is solved and the eigenvalues are available.



## D

---

# Mathematical Foundations of Kernel Methods

## D.1 Introduction

*Mercer kernels* (or *positive definite kernels*) are the foundations of powerful machine learning algorithms called *kernel methods*. Mercer kernels project implicitly the data in a high-dimensional *feature space* by means of a nonlinear mapping. The kernel theory has been developed during the first four decades of the twentieth century by some of the most brilliant mathematicians of the time. The concept of positive definite kernel has been introduced by [10]. Later on, remarkable contributions have been provided by [12][19][15][16][17][2][3]. In machine learning, the use of kernel functions to make computations, has been introduced by [1] in 1964. In 1995 a learning algorithm, *support vector machine (SVM)* [5] was introduced. SVM (see Chapter 9) uses Mercer kernels, as a preprocessing, to empower a linear classifier (*optimal hyperplane algorithm*) so as to make the classifier able to solve nonlinear tasks.

The aim of this appendix is to present an overview of the kernel theory, focusing on the theoretical aspects that are relevant for kernel methods (see Chapter 9), such as the Mercer kernels and the reproducing kernel Hilbert spaces.

The appendix is organized as follows: in Section D.2 the definitions of scalar product, norm and metric are recalled; in Section D.3 positive definite functions and matrices are presented; Section D.4 is devoted to conditionate positive definite kernels and matrices; negative definite functions and matrices are described in Section D.5; Section D.6 presents the connections between negative and definite kernels; Section D.7 shows how a metric can be computed by means of a positive definite kernel; Section D.8 describes how a positive definite kernel can be represented by means of a Hilbert space. finally some conclusions are drawn in Section D.9.

## D.2 Scalar Products, Norms and Metrics

The aim of this section is to recall the concepts of inner product, norm and metric [14].

**Definition D.1.** Let  $X$  be a set. A **scalar product** (or **inner product**) is an application  $\cdot : X \times X \rightarrow \mathbb{R}$  satisfying the following conditions:

$$\begin{aligned}
 (a) \quad & y \cdot x = x \cdot y && \forall x, y \in X \\
 (b) \quad & (x + y) \cdot z = (x \cdot z) + (y \cdot z) && \forall x, y, z \in X \\
 (c) \quad & (\alpha x) \cdot y = \alpha(x \cdot y) && \forall x, y \in X \quad \forall \alpha \in \mathbb{R} \\
 (d) \quad & x \cdot x \geq 0 && \forall x \in X \\
 (e) \quad & x \cdot x = 0 && \iff x = 0 \\
 (f) \quad & x \cdot (y + z) = (x \cdot y) + (x \cdot z) && \forall x, y, z \in X
 \end{aligned}$$

Axioms (a) and (b) imply (f). Using axiom (d) it is possible to associate to the inner product a quadratic form  $\|\cdot\|$ , called the *norm*, such that:

$$\|x\|^2 = x \cdot x.$$

More generally, the norm can be defined in the following way.

**Definition D.2.** The seminorm  $\|\cdot\| : X \rightarrow \mathbb{R}$  is a function that has the following properties:

$$\begin{aligned}
 \|x\| &\geq 0 && \forall x \in X \\
 \|\alpha x\| &= |\alpha| \|x\| && \forall \alpha \in \mathbb{R} \quad \forall x \in X \\
 \|x + y\| &\leq \|x\| + \|y\| && \forall x, y \in X \\
 x = 0 &\implies \|x\| = 0 && \forall x \in X
 \end{aligned}$$

Besides, if

$$x = 0 \iff \|x\| = 0 \tag{D.1}$$

the function  $\|\cdot\| : X \rightarrow \mathbb{R}$  is called **norm**.

Norms and inner products are connected by the *Cauchy-Schwarz's inequality*:

$$|x \cdot y| \leq \|x\| \|y\|.$$

**Definition D.3.** Let  $X$  be a set. A function  $\rho : X \times X \rightarrow \mathbb{R}$  is called a **distance** on  $X$  if:

$$\begin{aligned}
 (a) \quad & \rho(x, y) \geq 0 && \forall x, y \in X \\
 (b) \quad & \rho(x, y) = \rho(y, x) && \forall x, y \in X \\
 (c) \quad & \rho(x, x) = 0 && \forall x \in X
 \end{aligned}$$

The  $(X, \rho)$  is called a **distance space**.

If  $\rho$  satisfies, in addition, the **triangle inequality**

$$(d) \rho(x, y) \leq \rho(x, z) + \rho(y, z) \quad \forall x, y, z \in X$$

then  $\rho$  is called a **semimetric** on  $X$ .

Besides, if

$$(e) \rho(x, y) = 0 \quad \Rightarrow \quad x = y$$

In this case  $(X, \rho)$  is called a **metric space**.

It is easy to show that the function  $\rho(x, y) = \|x - y\|$  is a *metric*. We conclude the section introducing the concept of  $L_p$  spaces.

**Definition D.4.** Consider countable sequences of real numbers and let  $1 \leq p < \infty$ . The  $L_p$  space is the set of sequences  $z = z_1, \dots, z_n, \dots$  such that

$$\|z\|_p = \left( \sum_{i=1}^{\infty} |z_i|^p \right)^{\frac{1}{p}} < \infty$$

### D.3 Positive Definite Kernels and Matrices

We now introduce the concept of positive definite matrices.

**Definition D.5.** A  $n \times n$  matrix  $A = (a_{jk})$ ,  $a_{jk} \in \mathbb{R}$ , is called a **positive definite matrix** iff<sup>1</sup>

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k a_{jk} \geq 0 \quad (\text{D.2})$$

for all  $n \in \mathbb{N}$ ,  $c_1, \dots, c_n \subseteq \mathbb{R}$ .

The basic properties of positive definite matrices are underlined by the following result.

**Theorem 17** *A matrix is positive definite iff is symmetric and has all eigenvalues non-negative.*

A matrix is called *strictly positive definite* if all eigenvalues are positive. The following result (*Sylvester's criterion*), whose proof is omitted, is a useful tool to establish if a matrix is strictly positive definite.

**Theorem 18** *Let  $A = (a_{jk})$  be a symmetric  $n \times n$  matrix.  $A$  is strictly positive definite iff*

$$\det(a_{jk})_{j,k \leq p} > 0 \quad p = 1, \dots, n$$

*i.e. all its minors have positive determinants.*

Now we introduce the concept of *positive definite kernels*.

---

<sup>1</sup> iff stands for *if and only if*.

**Definition D.6.** Let  $X$  be a nonempty set. A function  $\varphi : X \times X \rightarrow \mathbb{R}$  is called a **positive definite kernel** (or **Mercer kernel**) iff

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k \varphi(x_j, x_k) \geq 0$$

for all  $n \in \mathbb{N}$ ,  $x_1, \dots, x_n \subseteq X$  and  $c_1, \dots, c_n \subseteq \mathbb{R}$ .

The following result, which we do not prove, underlines the basic properties of positive definite matrices.

**Theorem 19** A kernel  $\varphi$  on  $X \times X$

- is positive definite iff is symmetric.
- is positive definite iff for every finite subset  $X_0 \subseteq X$  the restriction of  $\varphi$  to  $X_0 \times X_0$  is positive definite.

Besides, if  $\varphi$  is positive definite, then  $\varphi(x, x) \geq 0 \quad \forall x \in X$ .

An example of Mercer kernel is the *inner product*, as stated by the following corollary.

**Corollary 3** The inner product is a positive definite (Mercer) kernel.

*Proof*

Applying the properties of the inner product, we have:

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k x_j \cdot x_k = \sum_{j=1}^n c_j x_j \cdot \sum_{j=1}^n c_j x_j = \left\| \sum_{j=1}^n c_j x_j \right\|^2 \geq 0$$

For Mercer kernels an inequality analogous to Cauchy Schwarz's one holds, as stated by the following result.

**Theorem 20** For any positive definite kernel  $\varphi$  the following inequality holds

$$|\varphi(x, y)|^2 \leq \varphi(x, x)\varphi(y, y). \quad (\text{D.3})$$

*Proof*

Without losing generality, we consider the matrix

$$A = \begin{pmatrix} a & b \\ b & d \end{pmatrix}$$

where  $a, b, d \in \mathbb{R}$ . Then, for  $w, z \in \mathbb{R}$  we have:

$$\begin{aligned} (w \ z) \begin{pmatrix} a & b \\ b & d \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} &= aw^2 + 2bwz + dz^2 \\ &= a \left[ w + \frac{b}{a}z \right]^2 + \frac{z^2}{a} [ad - b^2] \quad (\forall a \neq 0) \end{aligned}$$

The matrix  $A$  is positive definite iff  $a \geq 0$ ,  $d \geq 0$  and



$$\det \begin{pmatrix} a & b \\ b & d \end{pmatrix} = ad - b^2 \geq 0$$

Therefore for any positive definite kernel  $\varphi$  we have

$$|\varphi(x, y)|^2 \leq \varphi(x, x)\varphi(y, y)$$

Since both sides of the inequality are positive, we get:

$$|\varphi(x, y)| \leq \sqrt{\varphi(x, x)}\sqrt{\varphi(y, y)} \tag{D.4}$$

□

If we define  $\|x\|_\varphi \triangleq \sqrt{\varphi(x, x)}$  a *pseudonorm*, the inequality (D.4) becomes

$$|\varphi(x, y)| \leq \|x\|_\varphi \|y\|_\varphi$$

that recalls the Cauchy Schwarz's inequality of the inner product.

The following remark underlines that  $\|x\|_\varphi$  is a pseudonorm.

**Remark 3**  $\|x\|_\varphi$  is not a norm, since  $x = 0$  does not imply  $\|x\|_\varphi = 0$ .

*Proof*

We consider the kernel  $\varphi(x, y) = \cos(x - y)$ ,  $x, y \in \mathbb{R}$ .  $\varphi$  is a Mercer kernel, since we have:

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n c_i c_j \cos(x_i - x_j) &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j [\cos(x_i) \cos(x_j) + \sin(x_i) \sin(x_j)] \\ &= \left[ \sum_{i=1}^n c_i \cos(x_i) \right]^2 + \left[ \sum_{i=1}^n c_i \sin(x_i) \right]^2 \\ &\geq 0 \end{aligned}$$

But  $\|x\|_\varphi = 1 \quad \forall x$ . □

Now we introduce the result that allows to use Mercer kernels **to make inner products**.

**Theorem 21** Let  $K$  be a symmetric function such that for all  $x, y \in X$ ,  $X \subseteq \mathbb{R}$

$$K(x, y) \triangleq \Phi(x) \cdot \Phi(y) \tag{D.5}$$

where  $\Phi : X \rightarrow F$  and  $F$ , which is a Hilbert space,<sup>2</sup> is called the **feature space**.

$K$  can be represented in terms of (D.5) iff  $K = (K(x_i, x_j))_{i,j=1}^n$  is semi definite positive, i.e.  $K$  is a Mercer kernel.

Besides,  $K$  defines an **explicit** mapping if  $\Phi$  is known, otherwise the mapping is **implicit**.

---

<sup>2</sup> see Section D.8.

*Proof*

We prove the proposition in the case of finite dimension space. Consider a space  $X = [x_1, \dots, x_n]$  and suppose that  $K(x, y)$  is a symmetric function on  $X$ . Consider the matrix  $K = (K(x_i, x_j))_{i,j=1}^n$ . Since  $K$  is symmetric, an orthogonal matrix  $V = [v_1, \dots, v_n]$  exists such that  $K = V\Lambda V^T$ , where  $\Lambda$  is a diagonal matrix that has, the eigenvalues  $\lambda_i$  of  $K$ , as elements, while  $v_i$  are the eigenvectors of  $K$ .

Now we consider the following mapping  $\Phi : X \rightarrow \mathbb{R}^n$

$$\Phi(x_i) \triangleq (\sqrt{\lambda_t} v_{ti})_{t=1}^n$$

We have:

$$\Phi(x_i) \cdot \Phi(x_j) = \sum_{t=1}^n \lambda_t v_{ti} v_{tj} = (V\Lambda V^T)_{ij} = K_{ij} = K(x_i, x_j).$$

The requirement that all the eigenvalues of  $K$  are non-negative descends from the definition of  $\Phi$  since the argument of the square root must be non-negative.  $\square$

For the sake of completeness, we cite Mercer's theorem <sup>3</sup> which is the generalization of the proposition D.5 for the infinite dimension spaces.

**Theorem 22** *Let  $X (X \subseteq \mathbb{R}^n)$  be a compact set. If  $K$  is a continuous symmetric function such that the operator  $T_K$ :*

$$(T_K f)(\cdot) = \int_X K(\cdot, x) f(x) dx \quad (D.6)$$

*is positive definite, i.e.*

$$\int_{X \times X} K(x, y) f(x) f(y) dx dy \geq 0 \quad \forall f \in L_2(X) \quad (D.7)$$

*then we can expand  $K(x, y)$  in a uniformly convergent series in terms of eigenfunctions  $\Phi_j \in L_2(X)$  and positive eigenvalues  $\lambda_j > 0$ ,*

$$K(x, y) = \sum_{j=1}^{\infty} \lambda_j \Phi_j(x) \Phi_j(y) \quad (D.8)$$

It is necessary to point out the following remark.

**Remark 4** *The condition (D.7) corresponds to the condition (D.4) of the definition of the Mercer kernels in the finite case.*

---

<sup>3</sup> The theorem was originally proven for  $X = [a, b]$ . In [7] the theorem was extended to general compact spaces.

Now we provide examples of Mercer kernels that define implicit and explicit mapping. The kernel  $K(x, y) = \cos(x - y)$ ,  $x, y \in \mathbb{R}$  defines an explicit mapping. Indeed, we have

$$K(x, y) = \cos(x - y) = \cos(x)\cos(y) + \sin(x)\sin(y)$$

that is the inner product in a Feature space  $F$  defined by the mapping  $\Phi : \mathbb{R} \rightarrow \mathbb{R}^2$

$$\Phi(x) = \begin{pmatrix} \cos(x) \\ \sin(x) \end{pmatrix}$$

On the contrary, the Gaussian <sup>4</sup>  $G = \exp(-\|x - y\|^2)$  is a case of a Mercer kernel with an implicit mapping, since  $\Phi$  is unknown. The possibility to use Mercer kernels in order to perform inner product makes their study quite important for computer science. In the rest of this section we will present methods to make Mercer kernels.

### D.3.1 How to Make a Mercer Kernel

The following theorem shows that Mercer kernels satisfy quite a number of properties.

**Theorem 23** *Let  $\varphi_1$  and  $\varphi_2$  be Mercer kernels respectively over  $X \times X$  and  $X \subseteq \mathbb{R}^n$ ,  $a \in \mathbb{R}^+$ ,  $\cdot$  and  $\otimes$  the inner and the tensor product, respectively.*

*Then the following functions are Mercer kernels:*

1.  $\varphi(x, z) = \varphi_1(x, z) + \varphi_2(x, z)$
2.  $\varphi(x, z) = a\varphi_1(x, z)$
3.  $\varphi(x, z) = \varphi_1(x, z) \cdot \varphi_2(x, z)$
4.  $\varphi(x, z) = \varphi_1(x, z) \otimes \varphi_2(x, z)$

*Proof*

The proofs of the first and the second properties are immediate.

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n c_i c_j [\varphi(x_i, x_j)] &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j \varphi_1(x_i, x_j) + \sum_{i=1}^n \sum_{j=1}^n c_i c_j \varphi_2(x_i, x_j) \geq 0 \\ \sum_{i=1}^n \sum_{j=1}^n c_i c_j [a\varphi(x_i, x_j)] &= a \sum_{i=1}^n \sum_{j=1}^n c_i c_j \varphi(x_i, x_j) \geq 0. \end{aligned}$$

Since the product of positive definite matrices is still positive definite, the third property is immediately proved.

The tensor product of two positive definite matrices is positive definite, since the eigenvalues of the product are all pairs of products of the eigenvalues of the two components.  $\square$

The following corollaries provide useful methods in order to make Mercer kernels.

---

<sup>4</sup> For the proof of the positive definiteness of the Gaussian see Corollary 9.

**Corollary 4** Let  $\varphi(x, y) : X \times X \rightarrow \mathbb{R}$  be positive definite. The following kernels are also positive definite:

1.  $K(x, y) = \sum_{i=0}^n a_i [\varphi(x, y)]^i \quad a_i \in \mathbb{R}^+$
2.  $K(x, y) = \exp(\varphi(x, y))$

*Proof*

The first property is an immediate consequence of the Theorem 23. Regarding the second item, the exponential can be represented as:

$$\exp(\varphi(x, y)) = 1 + \sum_{i=1}^{\infty} \frac{[\varphi(x, y)]^i}{i!}$$

and is a limit of linear combinations of Mercer kernels. Since Mercer kernels are closed under the pointwise limit, the item is proved.  $\square$

**Corollary 5** Let  $f(\cdot) : X \rightarrow \mathbb{R}$  be a function. Then  $\varphi(x, y) = f(x)f(y)$  is positive definite.

*Proof*

We have:

$$\sum_{i=1}^n \sum_{j=1}^n c_i c_j f(x_j) f(x_k) = \left( \sum_{i=1}^n c_i f(x_i) \right)^2 \geq 0$$

$\square$

The foregoing propositions are very useful to make new Mercer kernels by means of existing Mercer kernels. Nevertheless to prove that a kernel is positive definite is generally not a trivial task. The following propositions, that we do not prove, are useful criteria that allow to state if a kernel is positive definite.

**Theorem 24 (Bochner)** If  $K(x - y)$  is a continuous positive definite function, then there exists a bounded nondecreasing function  $V(u)$  such that  $K(x - y)$  is a Fourier Stieltjes transform of  $V(u)$ , that is:

$$K(x - y) = \int_{-\infty}^{\infty} e^{i(x-y)u} dV(u)$$

If the function  $K(x - y)$  satisfies this condition, then it is positive definite.

**Theorem 25 (Schoenberg)** Let us call a function  $F(u)$  completely monotonic on  $(0, \infty)$ , provided that it is in  $C^\infty(0, \infty)$  and satisfies the condition:

$$(-1)^n F^{(n)}(u) \geq 0 \quad u \in (0, \infty), \quad n = 0, 1, \dots$$

Then the function  $F(\|x - y\|)$  is positive definite iff  $F(\sqrt{\|x - y\|})$  is continuous and completely monotonic.

**Theorem 26 (Polya)** Any real, even, continuous function  $F(u)$  which is convex on  $(0, \infty)$ , i.e. satisfies  $F(\alpha u_1 + (1 - \alpha)u_2) \leq \alpha F(u_1) + (1 - \alpha)F(u_2)$  for all  $u_1, u_2$  and  $\alpha \in (0, 1)$ , is positive definite.

On the basis of these theorems, one can construct different Mercer kernels of the type  $K(x - y)$ .

### D.4 Conditionate Positive Definite Kernels and Matrices

Although the class of Mercer kernels is adequately populated, it can be useful to identify kernel functions that, although non-Mercer kernels, can be used, in similar way, to compute inner products. To this purpose we define the conditionate positive definite matrices and kernels [13].

**Definition D.7.** A  $n \times n$  matrix  $A = (a_{ij})$   $a_{ij} \in \mathbb{R}$  is called a **conditionate positive definite matrix of order  $r$**  if it has  $n - r$  non-negative eigenvalues.

**Definition D.8.** We call the kernel  $\varphi$  a **conditionate positive definite kernel of order  $r$**  iff is symmetric (i.e  $\varphi(x, y) = \varphi(y, x) \quad \forall x, y \in X$ ) and

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k \varphi(x_j, x_k) \geq 0$$

$\forall n \geq 2, x_1, \dots, x_n \subseteq X$  and  $c_1, \dots, c_n \subseteq \mathbb{R}$ , with  $\sum_{j=1}^n c_j P(x) = 0$  where  $P(x)$  is a polynomial of order  $r - 1$ .

Examples of conditionate positive kernels are <sup>5</sup>:

$$\begin{aligned} k(x, y) &= -\sqrt{\|x - y\|^2 + \alpha^2} & \alpha \in \mathbb{R} & \text{Hardy multiquadric } (r = 1) \\ k(x, y) &= \|x - y\|^2 \ln \|x - y\| & & \text{thin plate spline } (r = 2) \end{aligned}$$

As pointed out by [9] conditionally positive definite kernels are admissible for methods that use a kernel to make inner products. This is underlined by the following result.

**Theorem 27** If a conditionate positive definite kernel  $k(x, y)$  can be represented as  $k(x, y) \triangleq h(\|x - y\|^2)$ , then  $k(x, y)$  satisfies the Mercer condition (6).

*Proof*

In [8, 11] it was shown that conditionate positive definite kernels  $h(\|x - y\|^2)$  generate semi-norms and  $\|\cdot\|_h$  defined by:

---

<sup>5</sup> The conditionate positive definiteness of Hardy multiquadrics is shown in Corollary 7.

$$\|f\|_h^2 = \int h(\|x - y\|^2) f(x) f(y) dx dy \quad (\text{D.9})$$

Since  $\|f\|_h^2$  is a seminorm,  $\|f\|_h^2 \geq 0$ . Since the right side of (D.9) is the Mercer's condition for  $h(\|x - y\|^2)$ ,  $h(\|x - y\|^2)$  defines a scalar product in some feature space. Hence  $k(x, y)$  can be used to perform an inner product.  $\square$

This result enlarges remarkably the class of kernels, that can be used to perform inner products.

## D.5 Negative Definite Kernels and Matrices

We introduce the concept of negative definite matrices.

**Definition D.9.** A  $n \times n$  matrix  $A = (a_{ij})$   $a_{ij} \in \mathbb{R}$  is called a **negative definite matrix** iff

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k a_{jk} \leq 0 \quad (\text{D.10})$$

$\forall n > 2, c_1, \dots, c_n \subseteq \mathbb{R}$ .

Since the previous definition involves integers  $n > 2$ , it is necessary to point out that any  $1 \times 1$  matrix  $A = (a_{11})$  with  $a_{11} \in \mathbb{R}$  is called negative definite. The basic properties of negative definite matrices are underlined by the following result.

**Theorem 28** A matrix is negative definite iff is symmetric and has all eigenvalues  $\leq 0$ .

A matrix is called *strictly negative definite* if all eigenvalues are negative.

Now we introduce the concept of the *negative definite kernels*.

**Definition D.10.** We call the kernel  $\varphi$  a **negative definite kernel** iff is symmetric (i.e.  $\varphi(x, y) = \varphi(y, x) \forall x, y \in X$ ) and

$$\sum_{j=1}^n \sum_{k=1}^n c_j c_k \varphi(x_j, x_k) \leq 0$$

$\forall n \geq 2, x_1, \dots, x_n \subseteq X$  and  $c_1, \dots, c_n \subseteq \mathbb{R}$  with  $\sum_{j=1}^n c_j = 0$ .

In analogy with the positive definite kernel, the following result holds:

**Theorem 29** A kernel  $\varphi$  is negative definite iff for every finite subset  $X_0 \subseteq X$  the restriction of  $\varphi$  to  $X_0 \times X_0$  is negative definite.

An example of a negative definite kernel is the square of the Euclidean distance.

**Corollary 6** *The kernel  $\varphi(x, y) = \|x - y\|^2$  is negative definite.*

*Proof*

We have:

$$\begin{aligned} \sum_{i,j=1}^n c_i c_j \varphi(x_j, x_k) &= \sum_{i,j=1}^n c_i c_j \|x_i - x_j\|^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j [\|x_i\|^2 - 2(x_i \cdot x_j) + \|x_j\|^2] \\ &= \sum_{j=1}^n c_j \sum_{i=1}^n c_i \|x_i\|^2 + \sum_{i=1}^n c_i \sum_{j=1}^n c_j \|x_j\|^2 - 2 \sum_{i,j=1}^n c_i c_j (x_i \cdot x_k) \\ &= -2 \sum_{i=1}^n \sum_{j=1}^n c_i c_j (x_i \cdot x_k) \qquad \left( \text{since } \sum_{j=1}^n c_j = 0 \right) \\ &\leq 0 \end{aligned}$$

since the inner product is positive definite.  $\square$

Important properties of negative definite kernels are stated by the following lemma, whose proof [4] is omitted for the sake of brevity.

**Lemma D.11.** *If  $\psi : X \times X \rightarrow \mathbb{R}$  is negative definite and satisfies  $\psi(x, x) \geq 0 \forall x \in X$  then the following kernels are negative definite*

- $\psi^\alpha$  for  $0 < \alpha < 1$ .
- $\log(1 + \psi)$

Consequence of the lemma is that *Hardy multiquadrics* is a conditionate positive definite kernel.

**Corollary 7** *The Hardy multiquadrics  $-\sqrt{\alpha^2 + \|x - y\|^2}$  is a conditionate positive definite kernel of order 1, for  $\alpha \in \mathbb{R}$ .*

*Proof*

The kernel  $\psi(x, y) = \alpha^2 + \|x - y\|^2$  is negative definite,

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n c_i c_j [\alpha^2 + \|x_i - x_j\|^2] &= \alpha^2 \left( \sum_{i=1}^n c_i \right)^2 + \sum_{i=1}^n \sum_{j=1}^n c_i c_j \|x_i - x_j\|^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j \|x_i - x_j\|^2 \qquad \left( \text{since } \sum_{i=1}^n c_i = 0 \right) \\ &\leq 0 \end{aligned}$$

for Corollary 6.

Therefore, for the previous lemma,  $\varphi(x, y) = \psi(x, y)^{\frac{1}{2}}$  is still negative definite. Hence, the opposite of  $\varphi$ , i.e. the Hardy multiquadrics, is a conditionate positive definite kernel of order 1.  $\square$

One consequence of Lemma D.11 is the following fundamental result that characterizes negative definite kernels.

**Corollary 8** *The Euclidean distance is negative definite. More generally, the kernel  $\psi(x, y) = \|x - y\|^\alpha$  is negative definite for  $0 < \alpha \leq 2$ .*

*Proof*

The result is immediate consequence of Corollary 6 and Lemma D.11.  $\square$

## D.6 Relations Between Positive and Negative Definite Kernels

Positive and negative definite kernels are strictly connected. If  $K$  is positive definite then  $-K$  is negative definite. On the contrary, if  $K$  is negative definite, then  $-K$  is a *conditionate positive definite kernel of order 1*. Besides, positive and negative definite functions are related by the following lemma.

**Lemma D.12.** *Let  $X$  be a nonempty set,  $x_0 \in X$ , and let  $\psi : X \times X \rightarrow \mathbb{R}$  be a symmetric kernel. Put  $\varphi(x, y) := \psi(x, x_0) + \psi(y, x_0) - \psi(x, y) - \psi(x_0, y_0)$ . Then  $\varphi$  is positive definite iff  $\psi$  is negative definite.*

*If  $\psi(x_0, x_0) \geq 0$  and  $\varphi_0(x, y) := \psi(x, x_0) + \psi(y, x_0) - \psi(x, y)$ , then  $\varphi_0$  is positive definite iff  $\psi$  is negative definite.*

*Proof*

For  $c_1, \dots, c_n \in \mathbb{R}$ ,  $\sum_{j=1}^n c_j = 0$  and  $x_1, \dots, x_n \in X$  we have

$$\begin{aligned} \sum_{j=1}^n \sum_{i=1}^n c_i c_j \varphi(x_i, x_j) &= \sum_{j=1}^n \sum_{i=1}^n c_i c_j \varphi_0(x_i, x_j) \\ &= - \sum_{j=1}^n \sum_{i=1}^n c_i c_j \psi(x_i, x_j). \end{aligned}$$

Therefore positive definiteness of  $\varphi$  implies the negative definiteness of  $\psi$ .

On the other hand, suppose that  $\psi$  is negative definite. Let  $c_1, \dots, c_n \in \mathbb{R}$  and  $x_1, \dots, x_n \in X$ . We put  $c_0 = - \sum_{j=1}^n c_j = 0$ . Then



$$\begin{aligned}
 &0 \geq \sum_{j=0}^n \sum_{i=0}^n c_i c_j \psi(x_i, x_j) \\
 &0 \geq \sum_{j=1}^n \sum_{i=1}^n c_i c_j \psi(x_i, x_j) + \sum_{j=1}^n c_j c_0 \psi(x_j, x_0) + \sum_{i=1}^n c_i c_0 \psi(x_i, x_0) + \|c_0\|^2 \psi(x_0, x_0) \\
 &0 \geq \sum_{j=1}^n \sum_{i=1}^n c_i c_j [\psi(x_i, x_j) - \psi(x_j, x_0) - \psi(x_i, x_0) + \psi(x_0, x_0)] \\
 &0 \geq - \sum_{j=1}^n \sum_{i=1}^n c_i c_j \varphi(x_i, x_j)
 \end{aligned}$$

Hence  $\varphi$  is positive definite. Finally, if  $\psi(x_0, x_0) \geq 0$  then

$$\sum_{j=1}^n \sum_{i=1}^n c_i c_j \varphi_0(x_i, x_j) = \sum_{j=1}^n \sum_{i=1}^n c_i c_j [\varphi(x_i, x_j)] + \psi(x_0, x_0) \left( \sum_{j=1}^n c_j \right)^2 \geq 0$$

□

The following theorem is very important since it allows us to prove that the Gaussian kernel is positive definite.

**Theorem 30 (Schoenberg)** *Let  $X$  be a nonempty set and let  $\psi : X \times X \rightarrow \mathbb{R}$  be a kernel. Then  $\psi$  is negative definite iff  $\exp(-t\psi)$  is positive definite  $\forall t > 0$ .*

*Proof*

If  $\exp(-t\psi)$  is positive definite then  $1 - \exp(-t\psi)$  is negative definite

$$\begin{aligned}
 \sum_{i=1}^n \sum_{j=1}^n c_i c_j [1 - \exp(-t\psi)] &= \left( \sum_{i=1}^n c_i \right)^2 - \sum_{i=1}^n \sum_{j=1}^n c_i c_j \exp(-t\psi) \\
 &= - \sum_{i=1}^n \sum_{j=1}^n c_i c_j \exp(-t\psi) \quad \left( \text{since } \sum_{i=1}^n c_i = 0 \right) \\
 &\leq 0
 \end{aligned}$$

since  $\exp(-t\psi)$  is definite positive.

The negative definite is also the limit

$$\lim_{t \rightarrow 0^+} \frac{1}{t} (1 - \exp(-t\psi)) = \psi$$

On the other hand, suppose that  $\psi$  is negative definite. We show that for  $t = 1$ , the kernel  $\exp(-t\psi)$  is positive definite. We choose  $x_0 \in X$  and, for Lemma D.12, we have:

$$-\psi(x, y) = \varphi(x, y) - \psi(x, x_0) - \psi(y, x_0) + \psi(x_0, x)$$

where  $\varphi$  is positive definite. Since

$$\exp(-\psi(x, y)) = \exp(\varphi(x, y)) \exp(-\psi(x, x_0)) \exp(-\psi(y, x_0)) \exp(\psi(x_0, x))$$

we conclude that  $\exp(-\psi)$  is positive definite. The generic case  $\forall t > 0$ , can be derived for induction.  $\square$

An immediate consequence of the previous theorem is the following result.

**Corollary 9** *The Gaussian  $\exp(-\frac{\|x-y\|^2}{\sigma^2})$  is positive definite, for  $x, y \in \mathbb{R}^n$  and  $\sigma \in \mathbb{R}$ .*

*More generally,  $\psi(x, y) = \exp(-a\|x-y\|^\alpha)$ , with  $a > 0$  and  $0 < \alpha \leq 2$ , is positive definite.*

*Proof*

The kernel  $\|x-y\|^\alpha$  with  $0 < \alpha \leq 2$  is negative definite, as shown in Corollary 8. Therefore for Theorem 30 the Gaussian is positive definite.  $\square$

We conclude this section reporting, without proving them, the following results.

**Lemma D.13.** *A kernel  $\psi : X \times X \rightarrow \mathbb{R}$  is negative definite iff  $(t + \psi)^{-1}$  is positive definite  $\forall t > 0$ .*

**Theorem 31** *A kernel  $\psi : X \times X \rightarrow \mathbb{R}$  is negative definite iff its Laplace transform  $\mathbb{L}(t\psi)$*

$$\mathbb{L}(t\psi) = \int_0^{+\infty} \exp(-ts\psi) d\mu(s)$$

*is positive definite  $\forall t > 0$ .*

Consequence of the Lemma D.13, is the following result.

**Corollary 10** *Inverse Hardy multiquadrics  $\psi(x, y) = (\alpha^2 + \|x-y\|^2)^{-\frac{1}{2}}$ ,  $\alpha \in \mathbb{R}$  is positive definite.*

*Proof*

Since  $(\alpha^2 + \|x-y\|^2)^{-\frac{1}{2}}$  is definite negative (see Corollary 7), Inverse Hardy multiquadrics is definite positive for Lemma D.13.  $\square$

## D.7 Metric Computation by Mercer Kernels

In this section we show how to compute a metric by means of a Mercer kernel. Thanks to a fundamental result [15][16], it is possible to associate a metric to a kernel. In order to show that we consider, associated to a Mercer kernel  $K$ , the kernel  $d(x, y)$ :

$$d(x, y) \triangleq K(x, x) - 2K(x, y) + K(y, y).$$

The kernel  $d(x, y)$  is negative definite.

**Corollary 11** *If  $K(x, y)$  is positive definite then  $d(x, y)$  is negative definite. Besides,  $\sqrt{d(x, y)}$  is negative definite.*

*Proof*

We have:

$$\begin{aligned} \sum_{i,j=1}^n c_j c_i d(x_j, x_i) &= \sum_{i,j=1}^n c_j c_i [K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)] \\ &= \sum_{j=1}^n c_j \sum_{i=1}^n c_i K(x_i, x_i) - 2 \sum_{i,j=1}^n c_j c_i K(x_i, x_j) + \sum_{i=1}^n c_i \sum_{j=1}^n c_j K(x_j, x_j) \\ &= -2 \sum_{i=1}^n \sum_{j=1}^n c_j c_i K(x_i, x_j) \quad \left( \text{since } \sum_{j=1}^n c_j = 0 \right) \\ &\leq 0 \end{aligned}$$

since  $K$  is definite positive.

Now we show that  $d(x, y) \geq 0$

$$\begin{aligned} d(x, y) &= K(x, x) - 2K(x, x)K(y, y) + K(y, y) \\ &\geq K(x, x) - 2\sqrt{K(x, x)K(y, y)} + K(y, y) \\ &\geq \left[ \sqrt{K(x, x)} - \sqrt{K(y, y)} \right]^2 \\ &\geq 0. \end{aligned}$$

Hence, for Lemma D.11,  $\sqrt{d(x, y)}$  is negative definite.  $\square$

Now we introduce the result [15][16].

**Theorem 32 (Schoenberg)** *Let  $X$  be a nonempty set and  $\psi : X \times X \rightarrow \mathbb{R}$  be negative definite. Then there is a space  $H \subseteq \mathbb{R}^X$  and a mapping  $x \mapsto \varphi_x$  from  $X$  to  $H$  such that*

$$\psi(x, y) = \|\varphi_x - \varphi_y\|^2 + f(x) + f(y)$$

where  $f : X \rightarrow \mathbb{R}$ . The function  $f$  is non-negative whenever  $\psi$  is.

If  $\psi(x, x) = 0 \ \forall x \in X$  then  $f = 0$  and  $\sqrt{\psi}$  is a metric on  $X$ .

*Proof*

We fix some  $x_0 \in X$  and define

$$\varphi(x, y) = \frac{1}{2} [\psi(x, x_0) + \psi(y, x_0) - \psi(x, y) - \psi(x_0, y_0)]$$

which is positive definite for Lemma D.12. Let  $H$  be the associated space for  $\varphi$  and put  $\varphi_x(y) = \varphi(x, y)$ . Then

$$\begin{aligned} \|\varphi_x - \varphi_y\|^2 &= \varphi(x, x) + \varphi(y, y) - 2\varphi(x, y) \\ &= \psi(x, y) - \frac{1}{2} [\psi(x, x) + \psi(y, y)] \end{aligned}$$

By setting  $f(x) := \frac{1}{2}\psi(x, x)$  we have:

$$\psi(x, y) = \|\varphi_x - \varphi_y\|^2 + f(x) + f(y).$$

The other statements can be derived immediately.  $\square$

As pointed out by [6], the negative definiteness of the metric is a property of  $L_2$  spaces. Schoenberg's theorem can be reformulated in the following way:

**Theorem 33** *Let  $X$  be a  $L_2$  space. Then the kernel  $\psi : X \times X \rightarrow \mathbb{R}$  is negative definite iff  $\sqrt{\psi}$  is a metric.*

An immediate consequence of Schoenberg's theorem is the following result.

**Corollary 12** *Let  $K(x, y)$  be a positive definite kernel. Then the kernel*

$$\rho_K(x, y) \triangleq \sqrt{K(x, x) - 2K(x, y) + K(y, y)}$$

*is a metric.*

*Proof*

The kernel  $d(x, y) = K(x, x) - 2K(x, y) + K(y, y)$  is negative definite. Since  $d(x, x) = 0 \quad \forall x \in X$ , applying Theorem 1 we get that  $\rho_K(x, y) \triangleq \sqrt{d(x, y)}$  is a distance.  $\square$

Hence, it is always possible to compute a metric by means of a Mercer kernel, even if an implicit mapping is associated with the Mercer kernel. When an implicit mapping is associated to the kernel, it cannot compute the positions  $\Phi(x)$  e  $\Phi(y)$  in the feature space of two points  $x$  and  $y$ ; nevertheless it can compute their distance  $\rho_K(x, y)$  in the feature space. Finally, we conclude this section, providing metric examples that can be derived by Mercer kernels.

**Corollary 13** *The following kernels  $\rho : X \times X \rightarrow \mathbb{R}^+$*

- $\rho(x, y) = \sqrt{2 - 2 \exp(-\|x - y\|^\alpha)}$  with  $0 < \alpha < 2$
- $\rho(x, y) = \sqrt{(\|x\|^2 + 1)^n + (\|y\|^2 + 1)^n - 2(x \cdot y + 1)^n}$  with  $n \in \mathbb{N}$

*are metrics.*

*Proof*

Since  $(x \cdot y + 1)^n$  and  $\exp(-\|x - y\|^\alpha)$  with  $0 < \alpha < 2$  are Mercer kernels, the statement, by means of the Corollary 12, is immediate.  $\square$

## D.8 Hilbert Space Representation of Positive Definite Kernels

First, we recall some basic definitions in order to introduce the concept of *Hilbert space*.

**Definition D.14.** A set is a **linear space** (or **vector space**) if the addition and the multiplication by a scalar are defined on  $X$  such that,  $\forall x, y \in X$  and  $\alpha \in \mathbb{R}$

$$\begin{aligned} x + y &\in X \\ \alpha x &\in X \\ 1x &= x \\ 0x &= 0 \\ \alpha(x + y) &= \alpha x + \alpha y \end{aligned}$$

**Definition D.15.** A sequence  $x_n$  in a normed linear space<sup>6</sup> is said to be a **Cauchy sequence** if  $\|x_n - x_m\| \rightarrow 0$  for  $n, m \rightarrow \infty$ .

A space is said to be **complete** when every Cauchy sequence converges to an element of the space.

A complete normed linear space is called a **Banach space**.

A Banach space where an inner product can be defined is called a **Hilbert space**.

Now we pass to represent positive definite kernels in terms of a *reproducing kernel Hilbert space (RKHS)*.

Let  $X$  be a nonempty set and  $\varphi : X \times X \rightarrow \mathbb{R}$  be positive definite. Let  $H_0$  be the space the subspace of  $\mathbb{R}^X$  generated by the functions  $\{\varphi_x | x \in X\}$  where  $\varphi_x(y) = \varphi(x, y)$ .

If  $f = \sum_j c_j \varphi_{x_j}$  and  $g = \sum_i d_i \varphi_{y_i}$ , with  $f, g \in H_0$ , then

$$\sum_i d_i f(y_i) = \sum_{i,j} c_j d_i \varphi(x_j, y_i) = \sum_j c_j g(x_j) \tag{D.11}$$

The foregoing formula does not depend on the chosen representations of  $f$  and  $g$  and is denoted  $\langle f, g \rangle$ . Then the inner product  $\langle f, g \rangle = \sum_{i,j} c_i c_j \varphi(x_i, x_j) \geq 0$  since  $\varphi$  is definite positive. Besides, the form  $\langle \cdot, \cdot \rangle$  is linear in both arguments.

A consequence of (D.11) is the *reproducing property*

$$\begin{aligned} \langle f, \varphi_x \rangle &= \sum_j c_j \varphi(x_j, x) = f(x) & \forall f \in H_0 \quad \forall x \in X \\ \langle \varphi_x, \varphi_y \rangle &= \varphi(x, y) & \forall x, y \in X \end{aligned}$$

Moreover, using Cauchy Schwarz's inequality, we have:

$$\begin{aligned} \|\langle f, \varphi_x \rangle\|^2 &\leq \langle \varphi_x, \varphi_x \rangle \langle f, f \rangle \\ |f(x)|^2 &\leq \langle f, f \rangle \varphi(x, x) \end{aligned} \tag{D.12}$$

Therefore  $\langle f, f \rangle = 0 \iff f(x) = 0 \quad \forall x \in X$ .

<sup>6</sup> A *normed linear space* is a linear space where a norm function  $\|\cdot\| : X \rightarrow \mathbb{R}$  is defined that maps each element  $x \in X$  into  $\|x\|$ .

Hence, the form  $\langle \cdot, \cdot \rangle$  is an *inner product* and  $H_0$  is a *Pre-Hilbertian space*.<sup>7</sup>  $\mathbb{H}$ , the completion of  $H_0$ , is a *Hilbert space*, in which  $H_0$  is a dense subspace. The Hilbert function space  $\mathbb{H}$  is usually called the *reproducing kernel Hilbert space (RKHS)* associated to the Mercer kernel  $\varphi$ . Hence, the following result has been proved.

**Theorem 34** *Let  $\varphi : X \times x \rightarrow \mathbb{R}$  be a Mercer kernel.*

*Then there is a Hilbert space  $\mathbb{H} \subseteq \mathbb{R}^X$  and a mapping  $x \mapsto \varphi_x$  from  $X$  to  $\mathbb{H}$  such that*

$$\langle \varphi_x, \varphi_y \rangle = \varphi(x, y) \quad \forall x, y \in X$$

*i.e.  $\varphi$  for  $\mathbb{H}$  is the **reproducing kernel**.*

## D.9 Conclusions

In this appendix, the mathematical foundations of the Kernel methods have been reviewed focusing on the theoretical aspects which are relevant for Kernel methods. First we have reviewed Mercer kernels. Then we have described negative kernels underlining the connections between Mercer and negative kernels. We have also described how a positive definite kernel can be represented by means of a Hilbert space. We conclude the appendix providing some bibliographical remarks. Mercer kernel and RKHS are fully discussed in [3] which also represents a milestone in the kernel theory. A good introduction to the Mercer kernels, more accessible to less experienced readers, can be found in [4]. Finally, the reader can find some mathematical topics of the kernel theory discussed in some handbooks on Kernel methods, such as [18][20].

---

<sup>7</sup> A Pre-Hilbertian space is a normed, noncomplete space where an inner product is defined.

---

## References

1. M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.
2. N. Aronszajn. La theorie generale de noyaux reproduisants et ses applications. *Proc. Cambridge Philos. Soc.*, 39:133–153, 1944.
3. N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.
4. C. Berg, J.P.R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag, 1984.
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
6. M. Deza and M. Laurent. Measure aspects of cut polyhedra:  $l_1$ -embeddability and probability. Technical report, Departement de Mathematiques et d' Informatique, Ecole Normale Superieure, 1993.
7. N. Dunford and T. J. Schwarz. *Linear Operators Part II: Spectral Theory, Self Adjoint Operators in Hilbert Spaces*. John Wiley, 1963.
8. N. Dyn. Interpolation and approximation by radial and related functions. In *Approximation Theory*, pages 211–234. Academic Press, 1991.
9. F. Girosi. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical report, MIT, 1993.
10. D. Hilbert. Grundzüge einer allgemeinen theorie der linearen integralgleichungen. *Nachr. Göttinger Akad. Wiss. Math. Phys. Klasse*, 1:49–91, 1904.
11. W. R. Madych and S. A. Nelson. Multivariate interpolation and conditionally positive definite functions. *Mathematics of Computation*, 54:211–230, 1990.
12. J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Royal Soc.*, A209:415–446, 1909.
13. C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite,. *Constructive Approximation*, 2:11–22, 1986.
14. W. Rudin. *Real and Complex Analysis*. Mc Graw-Hill, 1966.
15. I. J. Schoenberg. Metric spaces and completely monotone functions. *Ann. of Math.*, 39:811–841, 1938.
16. I. J. Schoenberg. Metric spaces and positive definite functions. *Trans. Amer. Math. Soc.*, 44:522–536, 1938.

17. I. J. Schoenberg. Positive definite functions on spheres. *Duke. Math. J.*, 9:96–108, 1942.
18. B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.
19. I. Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. *J. Reine Angew. Math.*, 140:1–29, 1911.
20. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.



---

## Index

- A-Law compander 31, 33
- $C_p$  statistics 153
- $L_p$  space 415
- $N$ -grams 244, 264, 313
  - discounting 270
  - equivalence classes 265
  - history 265
  - parameters estimation 266
  - smoothing 270
- $\epsilon$ -Isomap 302
- $\epsilon$ -insensitive loss function 213
- $\mu$ -Law compander 31, 33
- $z$ -transform 397
  - properties 399
  - region of existence 398
  
- a posteriori probability 88
- A/D conversion 22
- AAC 35
- absolute threshold of hearing 36
- accuracy 327
- achromatic colors 61
- acoustic impedance 17, 20
- acoustic waves
  - energy 17
  - frequency 15
  - intensity 17
  - period 15
  - physics 15
  - pressure variations 15
  - propagation 16
  - source power 17
  - speed 17
  
- activation functions 163
- ADABOOST 189
- ADALINE 172
- adaptive boosting 190
- addition law
  - for arbitrary events 386
  - for conditional probabilities 388
  - for mutually exclusive events 384
- adjacency matrix 237
- Advanced Audio Coding 35
- affinity matrix 237
- agglomerative hierarchical clustering 136
- agglomerative methods 135
- AIC 153
- AIFF 33
- Akaike Information Criterion 153
- aliasing 24
- amplitude 15
- Angstrom 55
- annealed entropy 151
- approximations of negentropy 295
- arcing 189
- articulators 20
- articulators configuration 20
- Artificial Neural Networks 162
- artificial neurons 163
- asynchronous HMMs 263
- AU 33
- audio
  - acquisition 22
  - encoding 32
  - format 32

- storage 32
- time domain processing 38
- auditory channel 20
- auditory peripheral system 20
- autoassociative approach 289
- autocorrelation function 45
- average distortion 118
- average magnitude 41
- average value reduction 69
  
- B-frame 72
- B-VOP 74
- back-propagation 176
- bagging 189
- Banach space 429
- bankcheck reading 336
- Bark scale 22
- baseline JPEG algorithm 66
- basic colors 56
- basilar membrane 21
- Batch K-MEANS 120
- batch learning 180
- batch update 119
- Baum-Welch Algorithm 256
- Bayer's pattern 54
- Bayes classifier 89
- Bayes classifier optimality 89
- Bayes decision rule 88, 92
- Bayes discriminant error 142
- Bayes error 89
- Bayes formula 88
- Bayes problem 89
- Bayes risk 92
- Bayes Theorem 88
- Bayesian Information Criterion 153, 154
- Bayesian learning 221
- Bayesian Theory of Decision 85
- Bayesian voting 188
- best approximation property 174
- bias 140
- bias-variance dilemma 140, 141
- bias-variance trade-off 141
- BIC 153, 154
- Bidirectional Frame 72
- Bidirectional VOP 74
- bigram 330
- binary classification 92
- binary classifier 92
- binary code 66
- bit-rate 32
- blind source separation 291
- blue difference component 60
- blue-green 51
- Bochner theorem 420
- boosting 190
- bootstrap 189
- bootstrap aggregation 189
- bottleneck layer 290
- bottom-up strategy 67
- boundary bias term 142
- boundary error 142
- Bounded Support Vectors 230
- box-counting dimension 284
- Bregman methods 208
- brightness 57, 62
- BSS 291
- BTD 85
  
- Cambridge database 328
- camera movement 374
- capacity term 144
- Cauchy kernel 240
- Cauchy Schwarz's inequality 414
- Cauchy sequence 429
- CCA 300
- CCD 52
- Central Limit Theorem 293
- centroid 101
- cepstrum 321
- Chernoff's bound 150
- chroma 57, 62
- chromatic colors 61
- chromatic response functions 64
- chromaticity coordinates 58, 59
- chromaticity diagram 59
- chrominance 68
- chrominance components 60
- Chunking and Decomposition 207
- CIE 57
- CIE XYZ 57
- class 82, 86, 89
- class-conditional probability density function 86
- classification 82, 161
- classification learning 82
- classifier 82, 89, 143
- classifier complexity 144

- classifiers
  - combination 185
  - diversity 185
- cluster 110
- clustering 110, 374
- clustering algorithms 83
- clustering methods 110
- CMOS 52
- CMU-SLM toolkit 274
- CMY 57, 60
- cochlea 21, 36
- cocktail-party problem 290
- codebook 114, 182
- codevector 114
- codevectors 182
- coin tossing 383
- color gamut 59
- color interpolation 54
- color models 57
- color quantization 56
- color space 57
- Colorimetric models 57
- compact discs 32
- complete data likelihood 113
- complex exponentials 396
- complex numbers 395
  - conjugate 396
  - modulus 396
  - polar representation 396
  - standard representation 395
- compositor 74
- compression 117
- conditional optimization problem 197
- conditional risk 91
- conditionally positive definite 240
- conditionate positive definite kernel 421
- conditionate positive definite matrix 421
- cones 50
- confidence term 144
- conjugate gradient 206
- consistency 236
- consistency of ERM principle 150
- consistent model selection criterion 154
- Constrained Maximum 208
- convex objective function 199
- convex optimization problem 199
- coordinate chart 301
- coordinate patch 301
- cornea 50
- correlation dimension 284
- correlation integral 284
- covariance 97, 393
- covariance matrix 97, 393
- coverage 329
- critical band 21, 321
- critical frequency 24
- cross-entropy 181
- crossvalidated committees 189
- crossvalidation 156
- crystalline lens 50
- cumulative probability function 390
- curse of dimensionality 84, 280, 281
- Curvilinear Component Analysis 300
- d-dimensional Hausdorff measure 284
- DAG 213
- DAGSVM 213
- data 81, 87
- data dimensionality 282
- DCT 68
- DCT coefficients 69
- dead codevectors 121
- decibel scale 18
  - sound pressure level 18
- decision boundaries 95
- decision function 89
- decision regions 95
- decision rule 86, 91
- decoder 74
- decoding 119
- Delaunay triangulation 115
- deletion 326
- demapping layer 290
- demosaiicing 54
- dendrogram 135
- deterministic annealing 235
- dichotomizer 92
- die rolling 384
- diffeomorphism 301
- differentiable manifold 301
- differential entropy 294
- digital audio tapes 32
- digital camera 52
- digital rights management 75
- digital signal 38

- Digital Video 71
- Digital Video Broadcasting 71
- dimensionality reduction methods 84
- Dirichlet Tesselation 115
- discontinuity function 371
- Discrete Cosine Transform 68, 319, 403
- Discrete Fourier Transform 402
- discriminability 103
- discriminant function 94
- discriminant function rule 94
- discriminant functions 94
- dispersion 392
- distance space 414
- diversity 185
- divisive methods 135
- DRM 75
- DV 71
- DVB 71
  
- E-step 113
- ears 20
- Eckmann-Ruelle Inequality 286
- effective number of parameters 158
- eigenvalues 410
- eigenvectors 410
- embedded reestimation 323
- empirical average distortion 118
- Empirical Quantization Error 116
- Empirical Quantization Error in Feature Space 234
- empirical risk 143, 150
- Empirical Risk Minimization Principle 150
- encoding 119
- energy 41
- ensemble methods 185
  - ADABOOST 189
  - bagging 189
  - Bayesian voting 188
  - bootstrap aggregation 189
  - crossvalidated committees 189
  - error-correcting output code 191, 192
- entropy 151
- entropy coding 66
- entropy encoding 66
- entropy of the distribution 155
- Epanechnikov Kernel 240
  
- ERM Principle 150
- error 89
- error function 142
- error surface 178
  - global minima 179
  - local minima 179
- error-correcting output code 191
- estimation error 144, 281
- Euler equation 396
- events
  - complementary 384
  - disjoint 384
  - elementary 384
  - equivalent 384
  - exhaustive set 388
  - intersection 384
  - mutually exclusive 384
  - statistically dependent 389
  - statistically independent 388
  - union 384
- evidence 88
- Expectation-Maximization method 112
- Expected Distortion Error 116
- expected loss 91, 150
- Expected Quantization Error 116
- expected risk 143
- expected value of a function 96
- expected value of a variable 96
- exploratory projection pursuit 297
  
- farthest-neighbor cluster algorithm 136
- FastICA algorithm 297
- FCM 133
- feature extraction 280
- Feature Space 204, 227, 417
- Feature Space Codebook 234
- feature vector 86, 280
- features 86, 279
- Fermat optimization theorem 197
- field of view 53
- first choice multiplier 209
- first milestone of VC theory 151
- Fisher discriminant 223
- Fisher linear discriminant 224
- fixed length code 66
- focal length 53
- forest 67

- Fourier Transform 321, 401
  - region of existence 401
- fourth-order cumulant 293
- FOV 53
- fovea centralis 50
- fractal-based methods 283
- frame 70, 370
- front end 316, 317
- front-end 313
- Fukunaga-Olsen's algorithm 283
- function approximation theory 281
- function learning 82
- fundamental frequency 19
- Fuzzy C-Means 133
- fuzzy clustering 234
- fuzzy competitive learning 134
  
- Gaussian heat kernel 306
- Gaussian Mixture 248
  - parameters estimation 259
- Gaussian Processes 217, 221
- GCV 157
- General Topographic Mapping 129
- generalization error 143
- generalized correlation integral 285
- Generalized crossvalidation 156
- Generalized Linear Discriminants 171
- generalized Lloyd algorithm 120
- Generalized Portrait 202
- generalized Renyi dimension 284
- generative model 291
- geodetic distance 302
- Geometric distribution 106
- GIF 65
- glottal cycle 19
- glottis 18
- gradient descent 179
- Gram matrix 206
- Gram-Schmid orthogonalization 298
- graph cut problem 237
- graph Laplacian 305
- graylevel image 54
- grayscale image 54
- greedy algorithm 68
- growth function 152
- GTM 129
- GTM Toolbox 133
  
- Hamming window 319
  
- handwriting recognition 313
  - applications 336
  - front end 317
  - normalization 317
  - preprocessing 317
  - segmentation 318
  - subunits 318
- hardware oriented color models 57
- Hardy multiquadrics 421, 423
- Hausdorff dimension 283
- HCV 57, 62
- Heaps Law 268
- Heaviside function 164
- Hertz 17
- Hidden Markov Models 244, 313
  - backward variable 254
  - continuous density 247
  - decoding problem 252
  - discrete 247
  - embedded reestimation 323
  - emission functions estimation 259
  - emission probability functions 247
  - ergodic 245
  - flat initialization 323
  - forward variable 250
  - independence assumptions 247
  - initial states probability 258
  - initial states probability estimation 258
  - learning problem 256
  - left-right 245
  - likelihood problem 249
  - parameters initialization 257, 323
  - state variables 245
  - three problems 248
  - topology 245
  - transition matrix 245
  - transition probabilities 245
  - transition probabilities estimation 258
  - trellis 250
  - variants 262
- hierarchical clustering 110, 134, 135
- Hilbert space 428, 429
- HIS 62
- histogram 374
- HLS 57
- HSB 57, 62, 63
- HSV 57, 62

- HTK 314, 321, 323
- hue 57, 60, 62
- hue coefficient functions 64
- Huffman coding 66, 155
- Huffman's algorithm 67
- Hybrid ANN/HMM models 263
- hyperbolic tangent 165
- Hyvarinen approximation of negentropy 295
- I-frame 72
- I-VOP 74
- i.i.d. 87
- IAM database 328
- ICA 290
- ICA model 291
- ICA Model Principle 293
- ill-posed problems 210
- image
  - histogram 374
- image file format standards 64
- image processing 49
- impulse response 40
- incomplete data 112
- incomplete data likelihood function 113
- independent and identically distributed 87
- Independent Component Analysis 290, 291
- independent components 291
- independent trials 383
- infinite VC dimension 153
- infomax principle 297
- inner product 414, 416, 419, 430
- Input Output HMMs 262
- insertion 326
- intensity 17, 61, 62
- International Telecommunications Union 31
- Intra VOP 74
- Intra-frame 72
- intrinsic dimensionality 129, 280, 282
- Inverse Hardy Multiquadrics 426
- iris 50
- Iris Data 107, 137, 159, 240, 308
- Isomap 302
- isomap 300
- isometric chart 302
- Isometric feature mapping 302
- Jensen inequality 199
- JND 56, 57
- JPEG 66
- just noticeable difference 56
- K-fold Crossvalidation 156
- K-Isomap 302
- k-means 379
- Karhunen-Loeve Transform 287
- Karush-Kuhn Tucker conditions 200
- Katz's discounting model 273
- Kernel Engineering 239
- Kernel Fisher Discriminant 223
- Kernel K-Means 234, 238
- Kernel Methods 413
- Kernel PCA 226
- Kernel Principal Component Analysis 226
- kernel property 212
- Kernel Ridge Regression 217
- kernel trick 195, 234
- keyframe 367
  - extraction 370, 378
- KKT conditions 203, 216
- Kolmogorov capacity 284
- KPCA 226
- kriging 221
- Kronecker delta function 130
- Kruskal's stress 299
- Kuhn Tucker conditions 200
- Kuhn Tucker Theorem 199
- Kullback-Leibler distance 296
- kurtosis 293
- Lagrange multipliers 198
- Lagrange Multipliers Method 197
- Lagrange's multipliers theorem 198
- Lagrange's stationary condition 198
- Lagrangian 198
- Lagrangian function 199
- Lagrangian SVM 209
- Laplacian Eigenmaps 300, 305
- large numbers
  - strong law of 384
- latent variable method 129
- latent variable model 291
- latent variables 291

- LBG algorithm 120
- learner 80
- learning by analogy 81
- learning from examples 81
- learning from instruction 81
- learning machine 81
- learning problem 81, 150
- learning rate 122, 180
- Learning Vector Quantization 182
- learning with a teacher 82
- leave-one-out crossvalidation 156
- leptokurtic 294
- letters 321
- lexicon 316, 321, 328
  - coverage 329
  - selection 328
- lightness 61
- likelihood 88
- likelihood ratio 93
- linear classifier 101
- Linear Discriminant Analysis 223
- Linear Discriminant Functions 101, 168
- Linear Programming 206
- linear space 429
- LLE 303
- Lloyd iteration 120
- local optimal decision 68
- Locally Linear Embedding 300, 303
- log-log plot 285
- logarithmic compander 30
- logistic sigmoid 165, 170
- long-wavelength 55
- loss function 90
- lossless compression 32
- lossy compression 32, 66
- lossy data compression 68
- loudness 17
- luminance 60
- LVQ\_PAK 184
  
- M-step 114
- machine learning 79
- macroblocks 72
- Mahalanobis distance 97
- Manhattan distance 125
- manifold 301
- manifold learning 300, 301
- manifold learning problem 301
  
- mapping layer 290
- Markov Models 244
  - independence assumptions 244
- Markov random walks 238
- masking 36
- mathematical expectation 391
  - linearity 391
- matrix 407
  - characteristic equation 411
  - determinants 408
  - eigenvalues 410
  - eigenvectors 410
- maximum likelihood algorithm 221
- maximum likelihood principle 180
- Maximum Likelihood problem 111
- MDL 155
- MDS 299
- MDSCAL 299
- mean of a variable 96
- mean value 391
  - linearity 391
- measure of non Gaussianity 293
- medium-wavelength 55
- Mel Frequency Cepstrum Coefficients 319
- Mel Scale 321
- Mel scale 22
- membership matrix 235
- Mercer kernel 416
- Mercer theorem 417
- method of surrogate data 286
- metric space 415
- MFCC 319
- microphone 23
- mid-riser quantizer 28
- mid-tread quantizer 28
- minimum algorithm 136
- Minimum Description Length 155
- minimum Mahalanobis distance
  - classifier 102
- minimum Mahalanobis distance rule 102
- minimum weighted path length 67
- minimum-distance classifier 101
- minimum-distance rule 101
- model assessment 145
- model complexity 143
- model selection 139, 145
- monochromatic image 55

- monochromatic primary 58
- moving average 39
- MPEG 34, 71
  - layers 34
- MPEG-1 71
- MPEG-2 71, 72
- MPEG-21 75
- MPEG-4 standard class library 73
- MPEG-4 terminal 73
- MPEG-7 75
- MPEG-7 description schemes 75
- Multiclass SVMs 212
- Multidimensional Scaling 299
- Multilayer networks 173
- MultiLayer Perceptron 168
- Multilayer Perceptron 174
- multivariate Gaussian density 96
- mutual information minimization 296
  
- nearest prototype classification 182
- nearest-neighbor cluster algorithm 136
- necessary and sufficient condition for consistency of ERM principle 152
- negative definite kernel 422
- negative definite matrix 422
- negentropy 294
- neighborhood graph 302
- Neural Computation 163
- Neural Gas 126
- Neural Networks 162
  - activation functions 163
  - architecture 166
  - bias 166
  - connections 166
  - layers 166
  - off-line learning 180
  - on-line learning 180
  - parameter space 178
  - weights 166
- Neurocomputing 163
- neurons 162
- Ng-Jordan algorithm 237
- nonlinear component 290
- Nonlinear PCA 290
- norm 414
- normal Gaussian density 96
- normalization 317
- normalized frequency 23
- normed linear space 429
- NTSC 60, 70
- NTSC color space 60
- Nyquist frequency 24
  
- o-v-o method 213
- o-v-r method 212
- observation sequence 244, 246
- Occam's razor 146
- One Class SVM 229, 235
- One Class SVM extension 235
- one-versus-one method 213
- one-versus-rest method 212
- Online K-MEANS 121
- online update 119
- operating characteristic 104
- optic chiasma 52
- optimal encoding tree 68
- Optimal Hyperplane 201, 202
- Optimal Hyperplane Algorithm 201, 413
- optimal quantizer 119
- Out-Of-Vocabulary words 316, 330
- oval window 21
  
- P-frame 72
- P-VOP 74
- PAL 61, 70
- Parallel Distributed Processing 163
- partitioning clustering 110
- pattern 86
- PCA 226, 287
- Perceptron 172
- perceptual coding 35
- perceptual quality 32
- perceptually uniform color models 61
- perplexity 265, 329
- PGM 65
- phase 15
- phonemes 321
- photopic vision 55
- Psychological color models 57
- Physiologically inspired models 57
- piece-wise linear function 165
- pinna 20
- pitch 17, 19
- pixel 53
- platykurtic 294



- PNG 65
- Polya theorem 421
- polychotomizer 92
- polytope 114
- poor learner 140
- Portable Bitmap 65
- portable graymap 65
- Portable Image File Formats 65
- Portable Network Map 65
- portable pixmap 65
- positive definite kernel 416
- positive definite matrix 415
- positive semidefinite matrix 97
- postal applications 336
- posterior 88
- Postscript 65
- PPM 65
- Pre-Hilbertian space 430
- precision 376
- Predicted VOP 74
- Predictive frame 72
- preprocessing 279, 317
- primal-dual interior point 206
- primary hues 61
- principal component 287
- Principal Component Analysis 226, 287, 379
- principal component analysis 98
- principal components 99
- prior 88
- prior probability 86, 88
- Probabilistic Finite State Machines 244
- probability
  - conditional 387
  - definition of 384
- probability density 389
- probability density function 95
- probability distribution
  - joint 390
- probability distributions
  - definition of 389
- probability of error 89
- projection indices 297
- prototype-based classifier 162, 182
- prototyped-based clustering 110
- Pulse Code Modulation 28
- pupil 50
- pure colors 57
- quadratic loss 180
- Quadratic Programming 205
- quantization 27
  - error 28, 30
  - linear 27
  - logarithmic 30
- Quantization table 69
- quantizer 117, 118
  - optimal 118
- radiance function 56
- random point 390
- random variables
  - continuous 389
  - definition of 389
  - discrete 389
- recall 376
- Receiver Operating Characteristic Curve 104
- recognition process 316
- red difference component 60
- regression 82, 161
- regularization constant 205
- regularization costant 210
- reinforcement learning 81, 82
- rejection 90
- relative frequency 383
- reproducing kernel 430
- reproducing kernel Hilbert space 211, 429, 430
- reproducing property 429
- retina 50
- retinal array 50
- retinotopic map 124
- RGB 57
- RGB image 59
- RGB model 57, 59
- ridge regression 217
- risk 91
- RKHS 211, 429, 430
- robust clustering algorithm 137
- ROC curve 104
- rods 51
- rote learning 80
- row-action methods 208
- saccadic 50
- Sammon's mapping 299
- sample space 384

- sampling 23
  - frequency 23
  - period 23
- sampling theorem 25
- saturation 57, 60–62
- saturation coefficient functions 64
- scalar product 414
- Schoenberg theorem 420, 425, 426
- Schwartz criterion 154
- scotopic light 51
- SECAM 61
- second choice multiplier 209
- second milestone of VC theory 152
- Self Organizing Feature Map 124
- Self-Organizing Map 124
- semimetric 415
- sensor resolution 53
- sensor size 53
- sequential data 243
- Shannon frequency 24
- Shannon's theorem 155
- shattered set of points 146
- shattering coefficient 151
- Shi and Malik algorithm 238
- short term analysis 40
- short-wavelength 55
- shot 367
  - boundaries 376
  - boundary 368
- shot boundary 370
  - detection 370
- signal-to-noise Ratio 28
- simplex method 206
- single layer networks 168
- Singular Value Decomposition 287, 308
- slack variables 205
- slant 317
- Slater conditions 200
- slope 317
- SMO for classification 207
- SMO for One Class SVM 231
- smooth homomorphism 301
- smooth manifold 301
- SOCMF 58
- SOFM 124
- softmax function 181
- SOM 124
- SOM Toolbox 126
- SOM-PAK 126
- Spam Data 138, 159
- sparseness 267
- sparsest separating problem 206
- spatial redundance 71
- spatial resolution of image 53
- Spectral Clustering Methods 236
- spectral graph theory 305
- spectrogram 319
- speech production 18
- speech recognition 313
  - applications 336, 339
  - front end 319
- SRM 149
- standard observer color matching
  - functions 58
- state of nature 86
- state space models 264
- stationary point 197
- statistical independence 291, 388
- Statistical Language Modeling 244, 274
- Statistical Learning Theory 149
- statistically independence 97
- statistically independent components 290
- steepest gradient descent algorithm 122
- step function 165
- stress 299
- strong hue 61
- Structural Risk Minimization 149
- sub-Gaussian 294
- substitution 326
- subtractive primaries 60
- subunits 322
- sufficient condition for consistency of
  - ERM principle 151
- super-Gaussian 294
- supervised learning 81, 109, 161
- Support Vector Clustering 235
- Support Vector Machines 184, 195
- Support Vectors 203, 230
- SVC 235
- SVD 287
- SVM 195
- SVM construction 204
- SVM for classification 200
- SVM for Regression 213

- Sylvester's criterion 415
- symmetric loss function 93
- symmetric matrix 97
- synapses 162
- system
  - linear 38
  - LTI 39
  - time invariant 38
- Takens' method 285
- TDT-2 328
- teacher 80
- television law 60
- temporal redundancy 71
- tennis tournament method 213
- tensor product 419
- Test error 143
- test set 145
- the Munsell color space 61
- Theory of Regularization 210
- Thin Plate Spline 421
- third milestone of VC theory 152
- threshold function 165
- threshold of hearing 17
- TIFF 65
- topographic ordering 131
- topological dimension 282
- topological map 124
- Topology Representing Network 127
- topology-preserving map 124
- Torch 181
- Torchvision 377
- Training error 143
- training sample 82
- training set 82, 145
- trellis 250
- triangle inequality 414
- trigram 330
- tristimulus values 59
- Turing-Good counts 272
- Unconstrained Maximum 208
- uncorrelated components 287
- uncorrelatedness 292
- uniform color space 61
- unigram 330
- univariate Gaussian density 96
- univariate normal density 96
- universal approximation property 174
- unsaturated colors 61
- unsupervised learning 81, 83, 109
- unvoiced sounds 19
- user oriented color models 57
- user-oriented color models 61
- validation set 145
- value 61, 62
- Vapnik-Chervonenkis dimension 146, 153
- Vapnik-Chervonenkis Theory 149
- variable 294
- variable length code 67
- variance 140, 392
- variance of a variable 96
- variance term 142
- VC dimension 146, 152, 153
- VC entropy 151
- Vector Quantization 117
- vector space 429
- video 367
  - browsing 370
  - scenes 367
  - segmentation 367, 370
  - story 367
- video object layers 74
- video object planes 74
- video objects 74
- video sessions 74
- violet 51
- virtual primaries 58
- visual cortex 52
- Viterbi Algorithm 252
- vitreous humor 50
- vocal folds 18
- vocal tract 18
- vocing mechanism 18
- voiced sounds 19
- VOP 74
- Voronoi Region in Feature Space 234
- Voronoi Set in Feature Space 234
- Voronoi Tessellation 115
- voting strategy 213
- WAV 33
- wavelength 17
- weak hue 61
- weak learner 140
- Weber's law 56

- well-posed problem 210
- whitening 297
- whitening process 98
- Whitening Transformation 98
- window 40
  - hamming 40
  - length 41
  - rectangular 40
- Winner-takes-all 121, 212
- Wisconsin Breast Cancer Database 138, 308
- Word Error Rate 325
- Word Recognition Rate 325
- worst case approach 152
- XOR problem 171
- yellow-green 51
- YIQ 57, 60
- YUV 57, 60
- Zero Crossing Rate 44
- zero-one loss 143, 150
- zero-one loss function 93
- zig-zag scheme 69
- Zipf Law 268