Assignment 6
Extending splitting to the multiclass context
Training decision trees

# 1 Assignment for those who are achieving projects

## 1.1 Implementation of a tree

### 1.1.1 Organization of the storage

A tree is a structure containing `T` a row-vector and `D` a structure with some detailed information.

- `T` is a row vector with as many components as there are nodes. Each component indicates the column of the parent node. Considering a specific node. It can be a terminal node, then no other component has a value equal to the column number of that specific node. Or this specific node is not a terminal node, it has two children nodes, meaning two and only two components of this row vector have as values the column number of this specific node. Of these two children nodes, the one being the left most in `T` is the one addressed when a sample has passed the splitting test of this node. The other one is addressed when a sample has failed the test.
- `D` is an array of structures of the same size than `T`, there is a one-to-one connection between each structure of `D` and each component of `T`. Each structure has the following fields.
  - `isLeaf` indicates whether the node is a terminal node, that here is called a leaf, 1 if it is true and 0 if not.
  - `class` is a field that exists only if this node is a leaf. This field indicates the predicted class for this leaf.
  - `feature` is a field that exists only if this node is not a leaf. Split at this node is made upon the feature value that can be read at the corresponding column of `X`.
  - `lambda` is a field that exists only if this node is not a leaf. It is the threshold used for splitting data at this node.
  - `s` is a field that exists only if this node is not a leaf. It is equal to 1 when the successful samples are those having feature values below the threshold. It is equal to $-1$ when the successful samples are those having feature values above the threshold.

### 1.1.2 Displaying the tree

- `PNF_show_tree(tree);`
  It shows graphically the tree. The implementation makes use of the Matlab function `treeplot`. Here is an example of numbering the nodes on the graph.

```
treeplot(tree.T(tree.D,:));
count = size(tree.T,2);
[x,y] = treelayout(tree.T(tree.D,:));
x = x'; y = y';
name1 = cellstr(num2str((1:count)'));
text(x(:,1), y(:,1), name1, 'VerticalAlignment','bottom','HorizontalAlignment','right')
title({'Level Lines'},'FontSize',12,'FontName','Times New Roman');
```

  `PNF_show_tree(tree)` displays also the specific information related to each node.

### 1.1.3 Raising the tree

- `[tree,n]=PNF_createRoot();`
  It creates a tree with only one node. `n` is an index pointing to this node.
- `[tree,n1,n2]=add2nodes(tree,n);`
  At node `n`, this function adds two nodes that are the children of this node. The added nodes are located at columns `n1` and `n2` of `tree.T`, with $n_1 \leq n_2$, meaning that node `n1` is addressed when the test is passed.
- `tree=PNF_fillNode(tree,n,feature,lambda,s);`
  This function stores in `tree` the information claiming that the node pointed by `n` is a split and not a leaf. The parameter values of this split are `feature`, `lambda` and `s`.
- `tree=PNF_fillLeaf(tree,n,class);`
  This function stores in `tree` that the node pointed by `n` is a leaf and not a split. It has only one parameter value, the class.

## 1.2 Finding the leaf associated to a sample

- `n=PNF_findRoot(tree,n);`
  This function finds the node that is the root of the tree.

- `[test,c]=PNF_isLeaf(tree,n);`
  `n` is an positive integer indicating the column of `tree.T` that is a specific node of the tree. The function returns `test=1` if this node is a leaf and `0` if not. If this node is a leaf, then `c` is the class indicated in the tree for this leaf.

- `n_new=PNF_child_of(tree,n,x);`
  This function should be used only if we know that node `n` is not a leaf. It tests whether sample `x` passes the test defined by the parameters associated to node `n`. If so then the new node `n_new` is the left most column of `tree.T` whose parent is `n`. Conversely if not, then the new node `n_new` is the right most column of `tree.T` whose parent is `n`.

## 1.3 Training a tree

- `info=PNF_train1();` This function builds randomly a tree. Once the tree is built, the class assignment of each leaf is set according to the dataset upon the majority vote rule.

$$c_l = \underset{c\in\{1,\dots,C\}}{\arg\max} \sum_n \mathbf{1}(y_n = c)\mathbf{1}(T_l(\mathbf{x}_n)) \tag{1}$$

  where $y_n$ is the true class to which sample $\mathbf{x}_n$ belongs and $\mathbf{x}_n$ are all training samples; $T_l(\mathbf{x})$ is true if sample $\mathbf{x}$ has satisfied all tests associated to all nodes joining the root to the leaf $l$.

- `y_hat=PNF_predict1(info,x);`

$$\widehat{y}_n = \sum_l c_l \mathbf{1}(T_l(\mathbf{x}_n)) \tag{2}$$

  where $l$ are all leaves of a tree.

  Equation (2) suggests to scan all leaves of the tree and for each leaf test if $\mathbf{x}$ passes all corresponding test and if so return the label $c_l$. A more straightforward algorithm is possible:

  1. Find the root of the tree.
  2. Return the class if the considered node is a leaf.
  3. Based on the sample `x` and the considered node, find the correct child node.
  4. Go back to step 2.

- `score=PNF_score1(info,name_of_file,predict_function);`
  This function generalizes in the multiclass context `PNC_score1`, actually this should have been done during the second assignment.

- `info=PNF_train2();`
  This function is similar to `PNC_train2` or to `PND_train2`, it uses `PNF_train1` many times to consider a large number of randomly designed trees and for each tree, it uses `score=PNF_score1` to select the tree having the best performance, here *best* means have the highest overall accuracy.

- `tree=PNF_train3_c_n_l(tree,n);`
  This function is needed for `PNF_train3`. It adds a new field at each node called `n_l` which lists the samples in the `X`-matrix of the training set that are considered at this node as they passed or failed all tests that are in the pathway leading to this node, not including the specific test at this node. It assumes that `n_l` is already computed in the parent node.

- `OS=c_OS(tree,n,method);`
  This function computes at node `n` the opportunity of splitting this node. Depending on `method`, this opportunity of splitting can the difference between the entropy at the node without being split and the average entropy once split. It can also the difference between the Gini-Index before the split and after the split. This opportunity of splitting this node is indicated in a new field added to each node called `OS` whose value is this opportunity of splitting.

- `info=PNF_train3(method,displaying_function);`
  This function builds a tree using entropy or Gini-index as defined in `method`. Its pseudo-code is the following.

  1. Create a tree with only one node.
  2. Find the samples considered at this node.
  3. Measure the opportunity of splitting this node.

4. Find within the whole tree, the node having the highest opportunity of being split.
5. If all nodes have a negative opportunity of being split, assign to each node the majority class, and return the tree.
6. If not, achieve a split at the node having the greatest opportunity and create node `n1` and `n2`.
7. Find samples remaining at node `n1` and at node `n2`.
8. Compute the opportunity of splitting nodes `n1` and `n2`.
9. Run `displaying_function`.
10. Go back to step 4.

`displaying_function` is any function using tree as parameter and displaying some measurements or testing the tree on the testing dataset. The computed values could be stored in a global variable available for future use.

### 1.4 Presenting the results

The `.pdf` document is named `project_NF.pdf` and contains any relevant information.

1. Create a graphic showing how the overall accuracy evolve as the tree is being raised when tested first on the training set and then on the testing set. Find the best tree as measured using the testing set.
2. Reconstruct the predicted label testing and training images using the completely grown tree and using the tree having the best performance on the testing set.

## 2 Assignment for those who are reviewing projects

The goal is to build Matlab functions that achieve some basic checks on the data provided along each project. Two files are to be delivered.

The first file is a `.pdf` document. Its name is `reviewer` followed by a number and an `F` indicating that it refers to the second assignment. The first part of this document explains what is tested by each test. The second part explains for each project what has passed and what has failed with precise values showing the problem. The third part is optional, it explains what supplementary information you would request from the projects and how this information could provide more valuable testing.

The second file is a `.m` script having the same name, it runs successively the different functions contained in this file that do the different testings.

## 3 Discussion

Your task is first of all to read all projects and check `Progress`. You should write a single `.pdf` document, named `discussionF.pdf` discussing how all projects have undergone this first step, the difficulties that have been overcome and those that remain challenging issues. You should then express your opinion as to whether I should come back on some specific issues. You may also add some specific comments to a specific project on *Discussions* [1] and some specific questions on *Questions*. You are also expected to write in *Questions* the answers to all other questions.

[1] Comments should be most respectful as any work needs attention, and regardless of it being possibly wrong, it is going to be useful to get a better understanding. So there can be no shame in being wrong.