TP : Decision trees and ensemble methods

In the first cell, we provide four toy datasets for classification, featuring both linear and non-linear decision frontiers. Using decision trees, random forests, and AdaBoost, we will develop and analyze classifiers capable of handling non-linear decision boundaries.

We also provide in the second cell auxiliary code for plotting the decision boundaries of the trees, called `plot_tree`.

Impurity measures

1) For the four datasets, create decision trees using both the Gini and entropy criteria. Plot the score of each tree as a function of `max_depth`, ranging from 1 to 10.
2) For the best `max_depth`, plot the decision frontier of the best tree (the one with the highest score) for both impurity measures. Use the provided function `plot_tree`.
3) Select the best impurity measure and `max_depth` for dataset 2 (blobs) and compare the decision frontier with that of random forests using only the testing data.

Ensembles

4) Load the `diabetes`, `iris`, and `digits` datasets from `sklearn`. Perform a 5-fold cross-validation for each problem, considering whether they are regression tasks (using `RandomForestRegressor`) or classification tasks. Use the R^2 score for regression and accuracy for classification.
5) For the `diabetes` dataset, conduct a feature importance analysis. Among the various techniques available in `sklearn`, including random forest feature importance and permutation feature importance, select the one covered in class. Plot a bar chart showing the mean and standard deviation of feature importance values and comment on the results.

Regression Consider the following function and a sample generated from noisy observations of the real-valued function.

6) Using the whole dataset, train a regression tree for depths 2, 3, and 4, as well as for a random forest. Plot the prediction and comment on the shape of the predictions. Which impurity criterion is used here ?

AdaBoost Implement AdaBoost as seen in class.

7) Implement the **AdaBoost** algorithm as seen in class. You can use `my_stump`, described in Question 13 or a **stump** from `sklearn`, i.e., a decision tree of `max_depth`=1.
   (a) Implement the `fit` function in the provided template in the notebook.
   (b) Run the code for 50 iterations for all the toy datasets. At each iteration, plot the result of the AdaBoost ensemble `ab` using `plot_tree(ab, X, y)`.
   (c) Plot the evolution of the loss and the misclassification rate on the training and test splits over the 20 iterations.

AUC ROC - AUC PR In this exercise we use the `circles` dataset. The first cell generates and plots the data.

8) Given the code template, complete the function to compute the `average_precision_score` and `precision_recall_curve` from `sklearn.metrics`. Plot the ROC and PR curves for the balanced dataset.
9) Train a random forest with 100 trees on the training split. Use `predict_proba` to obtain class membership probabilities for the test split.
10) Discard a percentage of the positive data using the `subsample_data` function provided and plot the ROC and PR curves again. Comment on the differences.
11) Express the precision, recall, false positive rate, and true positive rate as conditional probabilities on the true class labels Y and predicted class labels $\widehat{Y}$.
12) Express the F1-score as the harmonic mean of two of the above quantities.

Bonus - stump Now, as a bonus question, you are asked to implement a **stump** from scratch, i.e., a decision tree of `max_depth`=1. The tree should be able to handle both weighted and unweighted samples.

To verify its correctness, you can compare the learned function with a decision tree of depth 1. This stump can be used as a weak learner in AdaBoost in the previous exercise. We will only consider datasets where there are exactly two features and the class is binary, as in the four toy datasets given at the beginning.

13) Implement and test the following functions for the stump.

    (a) Implement the `fit` method : Since the weak learners are potentially executed a large number of times, efficiency is crucial. Use the incremental evaluation of the partitions : the complexity should be $O(ndc)$ instead of the naive $O(n^2dc)$ version. [1] (Note : a non-incremental version will be graded with half the points). Iterate in the 2-dimensions for every possible split, evaluate the quality of each split using an incremental version of the Gini index (next question) and store the best split.

    (b) Implement the `gini` method : Implement the Gini impurity coefficient for the case in which there are only 2 classes. In class we saw the unweighted case. As a recap, let C be the number if different classes, $p_k(S)$ be the ratio of datapoints of class k in region S. Then, the Gini index $G(S)$ is

    $$G(S) = 1 - \sum_{k=1}^{C} p_k(S)^2$$

    Given a split in which we have left and right regions S_r, S_l, let N_r (resp. N_l) the number of datapoints on S_r (resp. S_l). The Gini index of the split is the combination of the Gini of both regions,

    $$\frac{N_r}{N_r + N_l} G(S_r) + \frac{N_l}{N_r + N_l} G(S_l)$$

    For the generalization to the weighted sample, let $w_k(S)$ be the sum of the weights of all data-points of class k in S. The Gini index is defined as follows :

    $$G(S) = 1 - \sum_{k=1}^{C} \left(\frac{w_k(S)}{\sum_{k=1}^{C} w_k(S)} \right)^2$$

    Given a split in which we have left and right regions S_r, S_l, let $W_r = \sum_{k=1}^{C} w_k(S_r)$ (resp. W_l) the total weight on S_r (resp. S_l). The Gini index of the partition is the combination of the Gini of both regions,

    $$\frac{W_r}{W_r + W_l} G(S_r) + \frac{W_l}{W_r + W_l} G(S_l)$$

    (c) Implement the `predict` method. The input is an array of n d-dimensional observations. The output is a `np.array` of length n. Once the `predict` method is coded, use the given function `plot_tree(my_stump,X,y)` to plot `my_stump` on `dataset[1]`.

    (d) Check the correctness of your proposal using the code provided in the notebook. The result should be the same as that using `DecisionTreeClassifier` in `sklearn` with maximum depth 1. Test if for the unweighted case (all weights are equal) and a weighted case (initialize the weights randomly).

1. We assume that the sorting operations such as `argsort` are free.