Assignment 4

Computer Science Department, University of Crete

MACHINE LEARNING - CS 577, Fall 2024

Deadline: Wednesday, 11/12/2024, 23:55 on e-learn (`https://elearn.uoc.gr`).

Deliverable files: Submit a zip/tar etc. file containing:

- a report in PDF with ALL answers to theoretical tasks, observations and figures produced by scripts.
- **all** Python script files written by you in the scope of the assignment. Please make sure that you included the necessary comments in your code!

Exercise 1 - Entropy and Decision Trees (Theoretical) [50 points]

You are working for a company and the policy is that you must open all emails because some of them might be from interested customers. You are having many emails daily. Some of them contain viruses/scams and are classified by your companions which have opened them (no other way to label). You are the only one having a functioning (not-infected with virus) computer. You have the following information to consider.

You know whether or not emails s1 to s4 (see Table 1) are safe (non-virus), but you do not know about s5 to s8 (the test set, see Table 2).

- What is the entropy of Virus at the root?
- Which attribute should you choose as the root of a decision tree?
- Build the decision tree to classify as virus or not.
- Classify the test set samples and report the accuracy.

	Body Length	Bold letters	susp. adress	Virus
s1	23	0	Yes	¬Virus
s2	18	1	No	¬Virus
s3	43	0	Yes	¬Virus
s4	68	0	No	Virus

Table 1: **Train Set**

	Body Length	Bold letters	susp. adress	Virus
s5	20	1	Yes	Virus
s6	25	1	No	¬Virus
s7	60	0	Yes	Virus
s8	35	0	No	¬ Virus

Table 2: **Test Set**

Exercise 2 - Random Forest (Programming) [50 points]

▶ **Part A [30 points]**

You will have to implement the Random Forest classifier for continuous data. More specifically, you will have to implement the following 2 functions:

def TrainRF(X, Y, n_trees, min_samples_leaf)

Inputs:

- **X:** The sample data (samples along rows, features along columns).
- **Y:** The label vector. Y is the class variable you want to predict.
- **n_trees:** the number of trees to grow
- **min_samples_leaf:** minimum number of leaf node observations

Output:

- **model:** This model should contain everything required in order to classify new samples. It is up to you to decide the structure of the variable. The only requirement is that it is compatible with your next function.

def PredictRF(model, X)

Inputs:

- **model:** A model previously trained using TrainRF.

– **X:** A matrix of data to be classified (samples along rows, features along columns).

Output:

– **predictions:** The vector of predicted classes for each of the input samples.

Rules/Hints: You should use the function *sklearn.tree.DecisionTreeClassifier* (see the Documentation for more information) to create the individual trees. **Do not set a *random_state*** or else every tree will be the same.

The TrainRF function should use trees with a *'min_samples_leaf'* statically provided by user. The number of features that shall be considered at each node – as suggested by the literature – shall be $floor(sqrt(n_f))$, where n_f is the total number of features. Read carefully the **DecisionTreeClassifier** documentation as some of the necessary parameters can be calculated internally by using the right keyword. You can keep the other parameters to their default values.

NOTE: You **are not allowed to** solve this exercise by using existing implementations of the Random Forest classifier

See Sections 15.1, 15.2 and first paragraph of 15.3 in the book Elements of Statistical Learning for more details about Random Forests.

▶ Part B [20 points]

Test your implementations on the provided dataset: "*Dataset5_XY*". The last column of the dataset is the categorical label Y (it can only have values 0 or 1); the rest of the columns are the features X.

More specifically you will need to implement a script which will:

– Split the data into a training (70%) and a test sample (30%)

– Estimate the accuracies of each individual tree in the forest, and compare them against the one obtained using the prediction from the Random Forest. To do so, produce a histogram of the accuracies related to the individual trees, and indicate with a line or arrow its mean value, and with an other line/arrow the accuracy obtained with the Random Forest.

How does the discrepancy between the average tree accuracy and the forest accuracy changes between $min\_samples\_leaf = 1$ and 10, and why?

NOTE: Just for the sake of comparison, you are allowed to compare your results against an existing Random Forest algorithm to convince us that your code is doing the right job.

[Bonus 10 points]

Perform the proper comparison between the accuracy obtained from a basic **decision tree** (using all the features at each node, and no permutations), and your Random Forest algorithm. (Notice that above you were performing an *unfair* comparison, by checking against the trees in the forest, each using only some of the features)

The trees created by the **DecisionTreeClassifier** function are stochastic, so that every time you run them you will get different results. Therefore, comparing a **single** tree with the Random Forest has a small statistical power. You will have to check against the distribution of accuracies obtained, say, from 1 000 individual decision trees.

Create an histogram for the accuracies of the single decision trees, and, as above, use an line/arrow to indicate the histogram mean value and the accuracy of the Random Forest. Using this plot (data!), calculate the probability that if you used a single tree, you would obtain a similar or better accuracy than using the Random Forest.

NOTE1: To ensure randomness, do not forget to set *random_state=None* in $DecisionTreeClassifier$.

NOTE2: For this bonus part, you are allowed to modify the function calls. Adopt $min\_samples\_leaf = 1$.

Extra reading for practical understanding

In this assignment we do not request to implement the Decision Tree Classifier from scratch. However, if you want to dive deeper to the Decision Tree implementation, to comprehend the algorithm better, we propose this tutorial: `https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/` The second box of chapter **"3.3. Building a Tree"** (99 lines), gives all the necessary functions for the definition and the training of the Tree. The code is extremely well documented by the author (Jason Brownlee), so you can read this webpage for a full comprehension of the implementation behind the scenes. This Tree basically reproduces the algorithm discussed in class, and it uses the Gini cost function (as an alternative to the Information Gain) to evaluate the "pureness" of the nodes.