**Programming Assignment #3 : Decision Trees with Bagging**

The basic task is to build a complete working program in Python implementing the Decision Tree algorithm. You have to first develop the program and test it, using the *Bank Note Authentication (BNA) Dataset* [http://archive.ics.uci.edu/ml/datasets/banknote+authentication#](http://archive.ics.uci.edu/ml/datasets/banknote%2Bauthentication) from the UCI (Univ. California Irvine) Machine Learning Data Repository <https://archive.ics.uci.edu/ml/index.php>. The software that you develop should be independent of a specific dataset, and should be able to generate Decision Trees for any Classification data set. Hence you shall also apply the S/W developed above on the *Sensorless Drive Diagnosis (SDD) Data Set,* taken from the same repository [http://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis#](http://archive.ics.uci.edu/ml/datasets/Dataset%2Bfor%2BSensorless%2BDrive%2BDiagnosis).

Note the first data set using which you will develop your program is of medium size and has only two classes. The second is large, has 49 features, 12 classes and about 60 K samples hence you may like to use a smaller sample size (say 1 in 10 samples extracted randomly; or in a traverse over *count* where *count % 10 == 0* is extracted).

Further, you are to use the K-fold approach for creating bagged decision trees and then use the statistical mode for class identification for the test data. You shall compare the accuracy of test data for both the datasets between the original decision trees and the bagged decision trees.

You are to follow the step-by-step approach described below:

1. The Decision Tree algorithm addresses two levels of logic: the higher architectural level and the node level. Accordingly the algorithm that you design prior to actual coding should also map into these two levels. You must first work out the algorithm and only then start coding
2. The node level algorithm is the core set of operations of splitting the data arriving from above (assuming tree is top down) at each node based on the specific feature (i.e. parameter) and the specific (granulated) value of that feature that provides the max. information gain. The data is split into a right and left branch which then ends in corresponding nodes where the set of operations are repeated, till a terminal node (leaf) is reached. Most of these operations have been expressed as pseudo-code and provided in the slides given to you, along with other slides that described different aspects of the DT algorithm
3. A node is identified to be a terminal one if any one of these 3 criteria is met: (a) the depth of the node exceeds a maximum value that is a hyper parameter you will set (this hyper parameter needs to be modulated for different data sets); depth can be defined as the number of branches to the root node, (b) the number of data at the node falls below some threshold that is another user-set hyper parameter, or (c) all the samples at that node belong to the same class (hence splitting is redundant). For both (a) and (b) the statistical mode of the samples is used to identify the class of the terminal node
4. Next we address the logical aspects of the DT architecture. It would be obvious that the enveloping function (containing other functions) for the node-level calculations as you set out to create the tree starting from the root node (where all the training data is presented) right up to all the terminal nodes, will call itself recursively as the nodes are traversed. The approach will be analogous to other established algorithms called *depth first search* which you can see on the net, e.g. in <https://en.wikipedia.org/wiki/Depth-first_search>or in <https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>(among others). It is advisable that you familiarise yourself well with the DFS algorithm before you work out the details of the DT architecture-creation algorithm.
5. As stressed in #1, start creating code *only after you are sure about your algorithm*. Download the BNA dataset and split it into (approx.) 90-10 for training-testing; use this as the base data set for developing and testing your S/W. Ideally you should extract the 10% intermittently from the total data rather than in any contiguous block from it
6. Your code should be able to work under both training (i.e. tree development) and testing (i.e. given the features of some data sample, pass it though the DT and extract its class) modes. You will need to modify your code so that the same baseline program is able to handle both these modes
7. Download the SDD dataset and use it to validate your program: its generality and capability to work with diverse data sets with different levels of complexity. *Note that the tree structure will change with the dataset, but the tree-generation-program should be same*. That would close the first phase of development of your own DT S/W
8. Next, modify your developed code to work with bagging. You split the BNA dataset randomly into K parts, choose K = 2 to start. Train two separate DTs and verify their accuracy individually for the 10% test data. Then repeat for another random partitioning of the data, where K could be different (and odd number as well). In this way extract anywhere above 8 DTs, but an odd number
9. Pass the 10% test data through all the created DT’s, and extract the final class of each testing sample using the stat. mode of the classification obtained from all the DT’s. Evaluate the accuracy of the bagged DTs and compare with accuracy obtained from the single DT.
10. Your submission should be a folder containing your code, and a word doc containing comparison of your single- and bagged- DT accuracies for the BNA dataset, and the single-dataset accuracy of the SDD dataset. Importantly, you should provide information (in the doc) on the structure of your single DT for both the data sets, in terms of your selected hyper-parameters, tree depth, splitting feature and splitting-value-for-the-feature at each node, the class of each terminal node, the number of data sets at each terminal node, etc. An illustration would be ideal but not necessary (but see next point)
11. Visualization of your created decision trees (for both data sets) would be a logical completion of your work, and the visualized form is that in which a DT will be understood / interpreted / explained by the user. There are free S/W that enable visualization of DTs, see e.g. <https://mljar.com/blog/visualize-decision-tree/> or <https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc> for using *plot\_tree* or *Graphviz* (note here we use Information Gain and not Gini for best splitting); if you are able to create good visualization, you shall get **BONUS Marks** of 20% over the original.

**EXTRA BONUS**: Repeat Steps 8-10 and the BONUS step above for the SDD dataset. You may need to improve the generality (don’t confuse this with generalization ability of ML models) of your software so that it can create Bagged Decision Trees for any types of datasets. This will also give you the confidence that you now ‘own’ software that can serve practical and contemporary classification problems in the real world.

**Extra Marks**: 10% above base marks for this assignment.