Telecom-Paris

SD201 – DATA MINING

3rd Lab report

DECISION TREE

Written by:

TAHER ROMDHANE

SAIFEDDINE BARKIA

Supervised by:

MAURIO SOZIO

2020/2021

During this project we used Object-oriented programming to make a coherent code.

- We will explain the 2 classes that exists in **Node.py and DecisionTree.py**

  Our project is generalizable for any dataset that has categorical values.

We created two classes:

Class Node: In this class we stored all the necessary information that each node has to have :

- Left and right children of the nodes (they are initialized by None) if they don't have any.
- The type of the node: root, intermediate or leaf
- The level of the node in the tree
- The gini of the node
- The feature on which we split the data.
- The constraint on which we split the data. (Example : if feature = ' Sex' and constraint =1, then the left child of this node will have the observations that have 'Sex'< 1 (which means = 0) and the right child of this particular node will have the observation that have the attribute Sex>=1 (==1 in our case)
- Class nodes that are reserved for leaves nodes (we didn't make the class a class attributes, we added this attribute only for leaves.
- Index: an attribute that we have created to know the parent and the children of each node.

Class DecisionTree :

First of all we are going to define the attributes of this class:

- minNum: minimum number if node before it's considered by default as leaf node
- alpha : the parameter for the pruning to penalize the complexity of the model
- d: default value of the class

We created inside this class the functions that are demanded in the assignment. We will see them one by one.

Before we start, we assume that a tree is completely defined by its root.

All the utility methods are commented in the code.

We will focus on the main functions and we will try to explain them one by one

- _build_node(self,node,indices=None):
  In order to optimize the code and to maintain a light memory, in each split we are passing the indices instead of the whole data especially that this function is recursive. Each of the stopping conditions are well commented in the code.

In this function, we added a particular condition that will be useful for the post pruning part. Actually, we will save for each node the majority class so that when we can post prune, we know exactly what class we are going to assign to that new leaf node.

This method is the one called *BuildDecisionTree() in the assignment* , we changed its name because our class id called **DecisionTree**

- _generalization_error(self,root): this function calculate the generalization error of our tree. We are going to use this function to determine either or not we are going to post prune the tree.
- _post_prune(self):

The of this function is the following: we start to look for a node that has 2 leaf nodes as children and we create new tree without those two leaf nodes and we change the type of the parent node from 'intermediate' to 'leaf'. The class of this node is determined by the majority rule (that's the condition that we made in the _build_node function.)
For this new generated tree, we calculate again the generalization error and we compare it to the generalization error generated before this particular change.
If the new tree has better generalization error, we make the pruned one our tree and we continue looking for optimization. To ease the selection of the particular leaf, we used the 'priority queue' data structure on the level attribute. Actually, the priority queue is similar to a queue in which each element additionally has a priority associated with it. In our example, the level is our priority, so nodes with the highest level are stored at the top of the queue.
We continue the search of the optimization until, we have the best pruned tree with the lowest generalization error

- For **decision_functions.py :**
  This is the .py file that we are supposed to send. Since we have worked with Object oriented programming, we have imported the functions that exists in "**DecisionTree.py"** and used them to create the demanded functions.
- We have included also, as demanded, the 3 files, output_tree, generalization_error and postpruned_tree. Those files are generated everytime we run the functions.
- In the main function we have tested all the functions that we built during our projet.