Project on Decision Trees

The project consists of implementing an algorithm for building and postpruning a decision tree. Make sure to follow the instructions below, failure to do so might result in the failure of the project. In particular, you should implement the algorithm whose pseudocode has been provided in slides 84-86 of our lecture on decision trees. The input is stored in a file, one line per record with each attribute separated by a space. There are three attributes A, B, C. A, B are ordinal attributes (e.g. T-shirt size) with A taking values in $\{0, 1, 2, 3\}$, B taking values in $\{0, 1, 2\}$, C is the (binary) class taking values in $\{0, 1\}$. The attributes are stored in that order in the input file. You should build a binary decision tree where each node has two branches, except the leaves. The attribute used to split and the values for each split are those that maximize information gain. Recall that since we are dealing with ordinal attributes we do not want to have splits that violate the order (e.g. $0, 2, 3$ one one branch and 1 on the other). You should also implement the postpruning algorithm where the generalization error is defined as: number of training errors + $\alpha\cdot$ number of leaves.

You should implement three functions in three different files (each function gives the same number of points):

1. a function *BuildDecisionTree* which receives in input a file with the input data and the parameter $minNum$ (minimum number of recored per node), builds a decision tree (not yet pruned) and print it in output using the function "*print()*" (see instructions below).
2. a function that receives in input, the input data, a tree (in the format specified below) and a parameter α, computes the generalization error and print it in output using "*print()*"
3. a function that receives in input a tree, parameters α and $minNum$ and produces a pruned decision tree in the format specified below (using "*print()*").

Printing the Tree: We recall that the level of the root is 1, while the level of any node in the tree is equal to the level of its parent +1. You should print the tree starting from the root node in a breadth first search (BFS) fashion. After printing the root node, print all nodes at level 2 starting from the left child of the root, that is, the node which satisfies the feature constraint of the root node. Then print all nodes at level 3 while printing left children first and so on. For each node you should print the following information (one element per line) in this order:

1. whether it is the root, in which case you should print "Root", a leaf (print "Leaf"), or an intermediate node (print "Intermediate")
2. the string "Level " followed by the level of the node
3. the string "Feature " followed by the attribute name, e.g. "A" followed by a list of values of the attribute. Each element is separated by the space character. E.g. "Feature A 0 2 3". This lines specified the constraint on the feature. If the node is a leaf, skip this line.
4. the string "Gini " followed by the gini value of the records associated with that node. E.g. 0.2222 intermediate node (print)

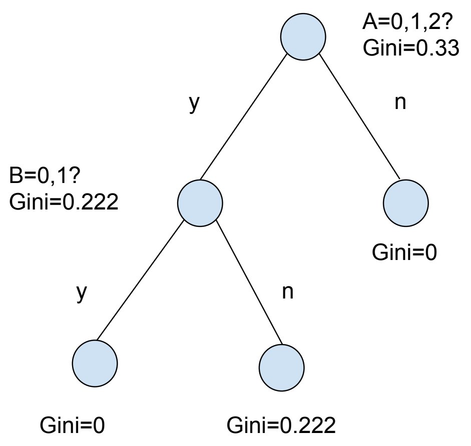

Figure 1: A simple decision tree.

For example, for the tree in Figure 1 you should print the following output, using the Python function "*print()*".

```
Root
Level 1
Feature A 0 1 2
Gini 0.33
Intermediate
Level 2
Feature B 0 1
Gini 0.2222
Leaf
Level 2
Gini 0
Leaf
Level 3
Gini 0
Leaf
Level 3
Gini 0.2222
```

What to send: You should send the python code of all three different functions in three different files. You should not send the notebook.