Thomas Huzij
tph2109@columbia.edu

Homework 3

1) Download the dataset cleve.txt. You need to transform it into an arff file to be able to use Weka. Make the necessary transformations to run the algorithms of this exercise.

See cleve.arff

2) Run Weka Experimenter with the following algorithms:

a. ADTrees
See results2.txt, key (1)
b. Random forests
See results2.txt, key (2)
c. Bagging on top of ADTrees
See results2.txt, key (3)
d. Decision stumps
See results2.txt, key (4)
e. C4.5 (J48 in Weka)
See results2.txt, key (5)

3) Build a table where you compare the 10 run 10 fold cross-validation test error, including their standard deviations with 10 iterations (for random forests use 100 trees. For decision stumps accept default values as you may not see an option to change the number of iterations).

Algorithm	**Test Error**	**Standard Deviation**
ADTree	19.54	5.93
RandomForest	17.89	6.65
Bagging (w/ ADTree)	17.08	7.12
DecisionStump	27.16	6.64
J48	22.25	6.72

The actual output from Weka can be found in files results3.arff and results3.txt. The file results3.arff can be loaded into Weka in the Analyze tab of the Experimenter for easy reading. Otherwise, results3.txt can be opened in any text editor and viewed as plain text.

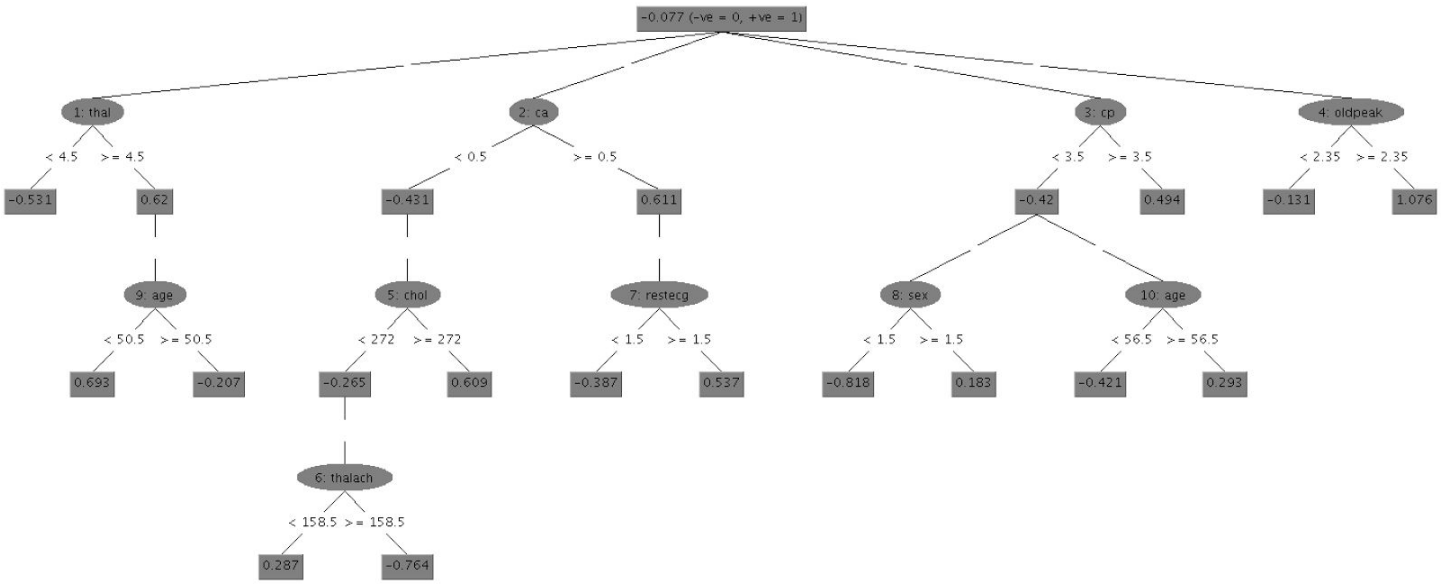
4) For C4.5, bagging and ADTree, extend the previous table running your experiment with 10, 50, and 100 iterations. For random forests, rerun the experiment changing the number of trees (try different numbers such as 100, 500, or 1000 trees or an optimal number that you can find). Also change the number of attributes to be used in random selection of random forests.

Algorithm	Test Error	Standard Deviation
ADTree (10 iterations)	19.54	5.93
ADTree (50 iterations)	21.33	6.48
ADTree (100 iterations)	22.16	6.74
RandomForest (100 trees, 0 attributes)	17.89	6.65
RandomForest (500 trees, 0 attributes)	17.52	6.50
RandomForest (1000 trees, 0 attributes)	17.15	6.26
RandomForest (1000 trees, 1 attribute)	17.55	6.63
RandomForest (1000 trees, 2 attributes)	17.35	6.33
RandomForest (1000 trees, 4 attributes)	17.15	6.26
RandomForest (1000 trees, 8 attributes)	17.92	6.55
Bagging (10 iterations)	17.08	7.12
Bagging (50 iterations)	17.01	6.86
Bagging (100 iterations)	16.88	6.92
DecisionStump	27.16	6.64
J48 (10 iterations)	22.25	6.72
J48 (50 iterations)	22.25	6.72
J48 (100 iterations)	22.25	6.72

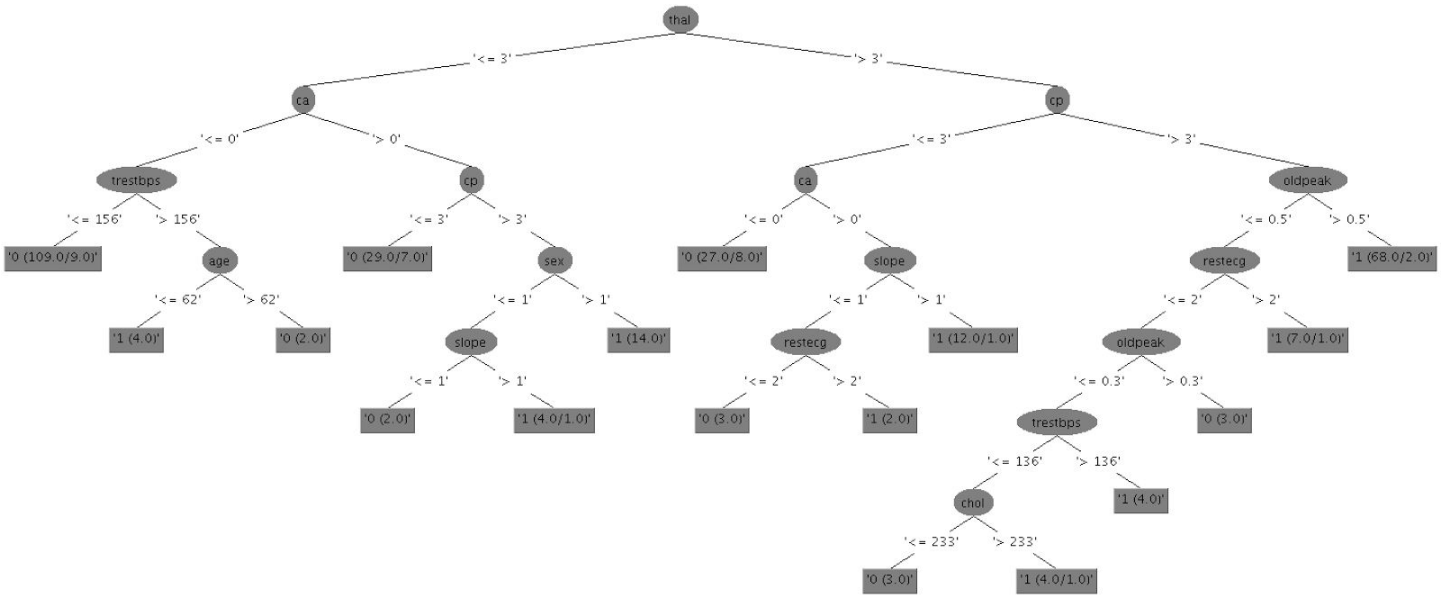
The actual output from Weka can be found in files results4_*.arff and results4_*.txt.

5) Using Weka explorer generate a sample tree for:

a. ADTree



b. C4.5



6) Write a report where include the above results and discuss your results:

a. Indicate how the different iterations, number of trees, and/or number of attributes affected your results.

Increasing the number of iterations increased the test error and the standard deviation for ADTree. On the other hand, increasing the number of iterations for Bagging actually decreased the test error. Additionally, in Bagging the standard deviation decreased at 50 iterations but increased slightly when the iterations were set to 100. Changing the number of iterations had no effect whatsoever on the test error or standard deviation for C4.5. Increasing the number of trees in RandomForest decreased both the test error and the standard deviation. Seeing that 1000 trees produced the best results for RandomForest, I set that as the constant tree count as I played with the number of attributes. At first, increasing the number of attributes actually hurt the results and both test error and standard deviation increased. However, with each new attribute the test error slowly dropped and at 4 attributes it finally returned to the same accuracy as the original 0 attribute test. Increasing the attribute count even further actually hurt the accuracy of RandomForest and the test error and standard deviation once again increased significantly.

b. Explain the difference between the different trees (graphs), and how you interpret them.

The ADTree graph consists of what are known as prediction nodes and decisions nodes. When traversing the tree, a prediction node conveys a value, which contributes to the final score of whether or not a decision should be made and/or a classification is successful. The decision nodes specify predicate conditions and decide which branch of the node should be taken next based on the value of the condition being compared. In an ADTree, both the root and all the leaf nodes are prediction nodes because at any given level of the tree, you should be able to determine a score for classification.

The C4.5 graph is similar in that it is also a classifier so it has prediction nodes and decision nodes. However, a major difference is that prediction nodes only appear in leaf nodes and nowhere else along the branches of a tree. As a result, when calculating a prediction in C4.5, only the value at the leaf is used to determine whether a classification is valid or not. This is in contrast to ADTree where each prediction node value that the algorithm encounters must be summed up to produce the final prediction. Whereas the final score calculation for an ADTree has to follow all prediction nodes that return true, a C4.5 tree traversal only goes down exactly one path through the tree. Additionally, the graph for C4.5 is much deeper and has many more levels than the graph for ADTree. This makes sense because an accurate prediction can only be made once enough entropy has been removed from the system and it takes several iterations to do so.

This can be interpreted to mean that C4.5 highly prioritizes the concept of information gain and relies on each new decision made as the tree is traversed to help narrow down whether a classification should be successful, while ADTree relies on the sum of all valid properties to create a better understanding of the instance to create a prediction about the class of that instance. This can result in ADTree performing with much higher accuracy because it uses an aggregation of tree traversals and predictions to make its choice while C4.5 only makes one final prediction after traversing the tree. This can be confirmed by looking at the corresponding test error values in the results.

c. Tell how you calculate the final score and how you will classify a specific instance in the case of ADTree.

To calculate the final score for an instance in ADTree, at each new decision node that is encountered, you must always take a branch if the decision node is true for that instance. However, if the decision node returns false, you still add the prediction value to the final score. Any prediction node value that is encountered along this traversal of the tree must be added to the final score of the prediction. Once the tree traversal is

completed, if the final score is positive then the instance is in the class and if the final score is negative then the instance is not in that class. Additionally, the magnitude of the final score tells us the confidence with which the classification prediction is being made. If given a specific instance in ADTree, start at the root and immediately add the value at the root to your final score because the root is a prediction node. Then for each of its children, follow the correct branch depending on the instance's value for those attributes. Whenever a leaf node is encountered, add the prediction value to the final score. Once there are no remaining branches to traverse, make a class prediction based on whether the final score is positive (it's in the class) or negative (it's not in the class).

d. Conclusions: what was the most adequate method under what criteria, and why.

It is clear from the data that the best algorithm to use for this dataset is RandomForest with 1000 trees and the default amount of attributes (0, but 4 attributes works as well because they returned the same test error). RandomTree minimized the test error which means it will be able to generate the most accurate predictions out of any of the algorithms, at least for this specific dataset. ADTree with only 10 iterations had the lowest standard deviation which means it exhibits less variance, but its test error was drastically higher so it's clear that RandomTree is preferable.