Thomas Huang
EECS 349 Problem Set #2

1. We defined a decision tree as a structure made up of node classes. Each node has a property that says on which attribute it will split. It also has a collection of branches, which are other nodes or leaves. A node also has a probability associated with the total wins over total instances in its remaining subset of the data. When travelling down a decision tree, this probability is the value that is returned when a terminating node is reached. The program will then use that probability to generate a prediction for an example (e.g. if the probability is 70%, there is a 70% chance the tree will predict a win). When the makeTree() function is called, the root node of the tree is returned.

2. Instances are grouped together in a dataset. During training, the learn function is called with a whole set and a root node, and the dataset is divided recursively into subsets that correspond to the various attributes. Each specific instance is represented as a list of floats.

3. The attribute for each node was chosen by splitting the node's associated subset of data by each possible attribute. The attribute whose split resulted in the most information gain (determined by weighted sums of resulting splits' entropies compared to the current node's entropy) was the chosen attribute.

4. Missing attributes were assigned a median of the current dataset in the function call. During training, if any win/loss classifications were missing, the instance was not used to train.

5. The termination criterion for the learning process were a maximum tree height, and also if all parts of a dataset split to one branch and none to another. The splitting restriction formed leaf nodes at necessary positions, and the maximum tree height makes sure that performance will be consistent. The default maximum tree height is 10.

6. See file dnfOutput.txt in .code directory

7. If the injured players on both teams is less than one, the opponent win percentage is less than 0.28, your win percentage is greater than 0.20, then your likelihood of winning is 0.886, or 88.6%

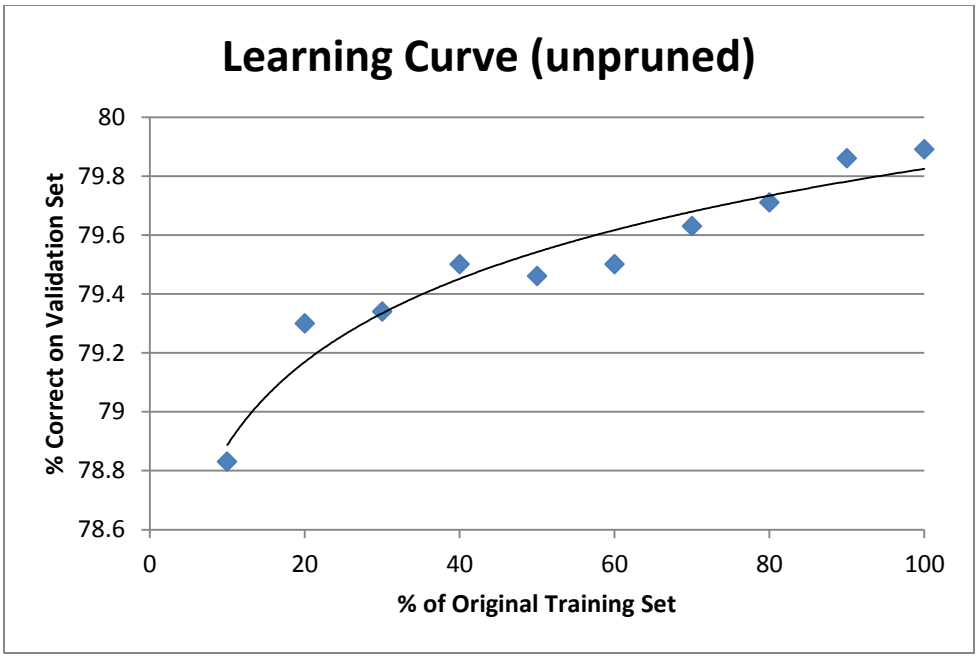
8. For our method of pruning, we looked at the best possible information gain (change in entropy) from one node to the branch nodes below it. If the information gain is below a minimal threshold value, there is no point in traversing below that node, so its branches are pruned.

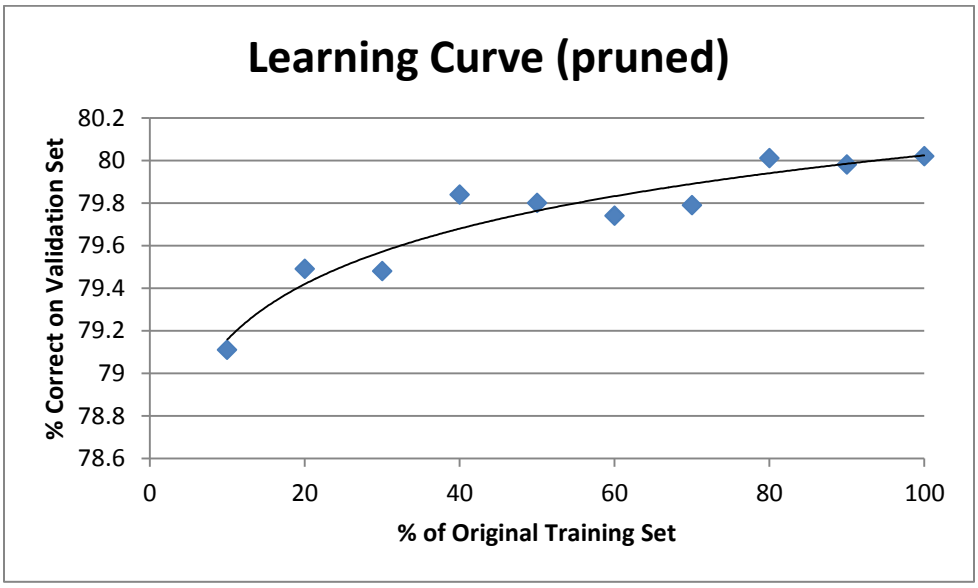
9. See file dnfOutput.txt in .code directory

10. The unpruned tree (with default settings) had 1742 splits, while the pruned tree with the same settings had 625 splits. Thus, the difference between them was 1117 splits.

11. With default settings, the unpruned tree had an accuracy of 79.67%. With the same settings, the pruned tree had an accuracy of 79.82%. We determined that this was a relatively negligible difference.

12. Below are the learning curve graphs for both the unpruned and pruned trees. We did not see any significant differences between the two graphs, but the pruned tree did have a flatter learning curve, and the value at 10% of the Training Set was lower for the pruned tree.





13. The pruned tree should perform better on the unlabeled test set. From running both trees on various sets, we noticed that the pruned tree had less variance in its accuracy.

14. We found out that the number of injured players for both teams and winrates for both teams, but run differential and opposing starting pitcher turned out to be poor fits.

15. Greg did the algorithm design and explained necessary concepts/ideas along with getting the DNF Boolean Formula, and Thomas did the actual Python implementation of the trees/functions.