**Student Name: Siyu Zhang**

**NetID: szw910**

**1. How did you handle missing attributes in examples**

We decided to plug the mode of the attribute for the missing attribute. That is, suppose some examples are missing values of attribute A. First we use training example anyway and sort through tree. Then assign most common value of A among other examples with same target value.

**2. Apply your algorithm to the training set, without pruning. Print out a Boolean formula in disjunctive normal form that corresponds to the *unpruned* tree learned from the training set. For the DNF assume that group label "1" refers to the positive examples. NOTE: if you find your tree is cumbersome to print in full, you may restrict your print-out to only 16 leaf nodes.**

(numinjured<1.0 ^ oppnuminjured<1.0 ^ opprundifferential<32.0) v (numinjured<1.0 ^ oppnuminjured<1.0 ^ opprundifferential>=32.0 ^ oppnuminjured<0.0) v (numinjured<1.0 ^ oppnuminjured<1.0 ^ opprundifferential>=32.0 ^ oppnuminjured>=0.0 ^ dayssincegame>=2.0) v (numinjured<1.0 ^ oppnuminjured>=1.0 ^ opprundifferential<43.0 ^ rundifferential>=71.0) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent<0.261062525 ^ winpercent<0.661899761) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525 ^ startingpitcher=1) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525 ^ startingpitcher=1 ^ startingpitcher=2) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525 ^ startingpitcher=1 ^ startingpitcher=2 ^ startingpitcher=3) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525 ^ startingpitcher=1 ^ startingpitcher=2 ^ startingpitcher=3 ^ startingpitcher=4) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525 ^ startingpitcher=1 ^ startingpitcher=2 ^ startingpitcher=3 ^ startingpitcher=4 ^ startingpitcher=5) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent>=0.354916694 ^ winpercent<0.777399225 ^ winpercent<0.219309677) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent>=0.354916694 ^ winpercent<0.777399225 ^ winpercent>=0.219309677) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent>=0.354916694 ^ winpercent>=0.777399225 ^ oppwinningpercent>=0.375600184) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured<2.0 ^ opprundifferential<13.0) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured<2.0 ^ opprundifferential>=13.0 ^ opprundifferential<32.0) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured>=2.0 ^ oppwinningpercent>=0.15431423 ^ oppwinningpercent<0.482567275) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured>=2.0 ^ oppwinningpercent>=0.15431423 ^ oppwinningpercent>=0.482567275)

**3. Explain in English one of the rules in this (unpruned) tree?**

The higher the winpercent is, the more likely 0 will win.

**4. How did you implement pruning?**

We use post-pruning. In post-pruning, we grow the tree full until all leaves are pure. Then we find sub-trees that cause overfitting and we prune them. From the initial labeled set, we set aside a pruning set, unused during training. For each sub-tree, we replace it by a leaf node labeled with the training instances covered by the sub-tree. If the leaf node does not perform worse than the sub-tree on the pruning et, we prune the sub-tree and keep the leaf node because the additional complexity of the sub-tree is not justified; otherwise, we keep the sub-tree.

**5. Apply your algorithm to the training set, with pruning. Print out a Boolean formula in disjunctive normal form that corresponds to the *pruned* tree learned from the training set.**

(numinjured<1.0 ^ oppnuminjured<1.0) v (numinjured<1.0 ^ oppnuminjured>=1.0 ^ opprundifferential<43.0 ^ rundifferential>=71.0) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent<0.261062525 ^ winpercent<0.661899761) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent<0.354916694 ^ oppwinningpercent>=0.261062525) v (numinjured>=1.0 ^ oppnuminjured<2.0 ^ oppwinningpercent>=0.354916694) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured<2.0 ^ opprundifferential<13.0) v (numinjured>=1.0 ^ oppnuminjured>=2.0 ^ numinjured>=2.0 ^ oppwinningpercent>=0.15431423)

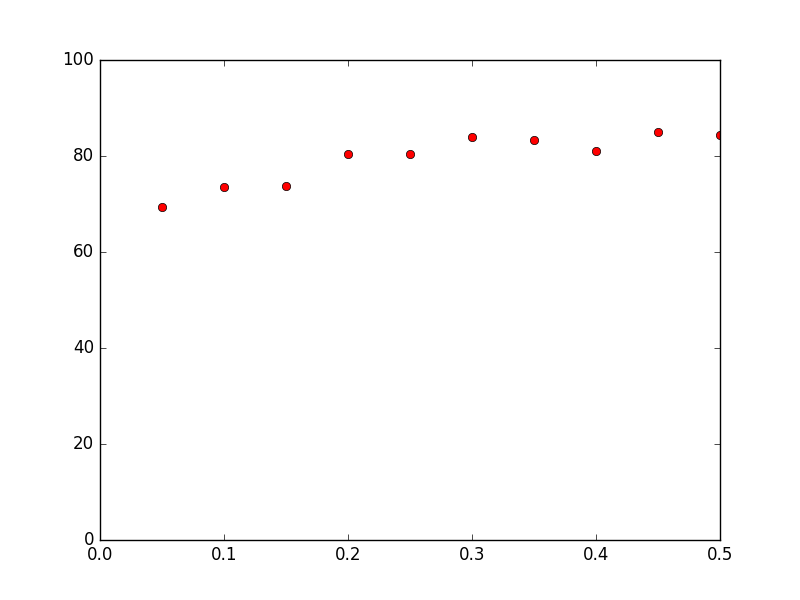
**6. What is the difference in size (number of splits) between the pruned and unpruned trees?**

The unpruned tree has 11 splits and the pruned tree has 21 splits.

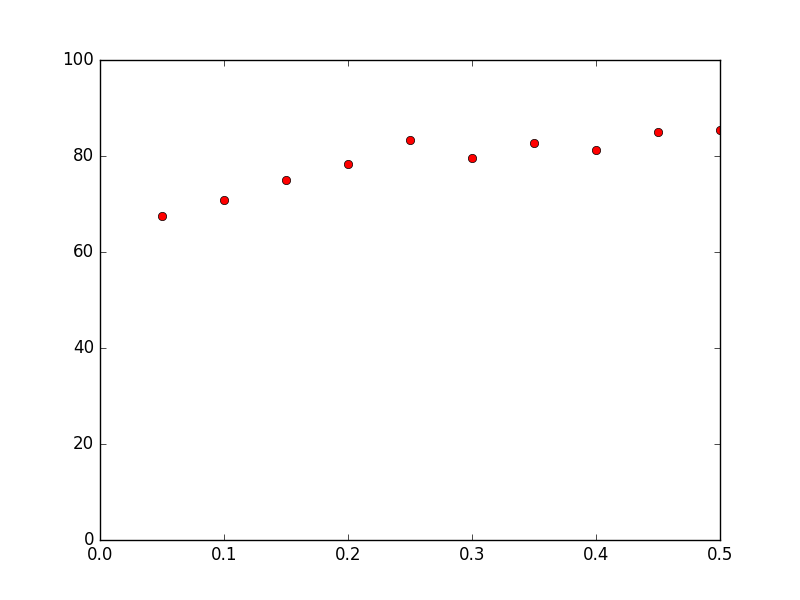
**7. Test the unpruned and pruned trees on the validation set. What are the accuracies of each tree? Explain the difference, if any.**

The accuracy of the unpruned tree is 85.2564102564% and the accuracy of the pruned tree is 86.1111111111%. The accuracy of the pruned tree is a little bit higher than the accuracy of unpruned tree because by pruning the decision tree, we reduce the risk of overfitting to the training data.

**8.Create learning curve graphs for both unpruned and pruned trees. Is there a difference between the two graphs?**



Learning curve graph for the unpruned tree



Learning curve graph for the pruned tree

There is some difference between the two learning curve graphs but not so significant. The pruned tree is relatively more smooth.

**9. Which tree do you think will perform better on the unlabeled test set? Why? Run this tree on the test file and submit your predictions as described in the submission instructions.**

I think that the pruned tree will perform better on the unlabeled test set. This is because by pruning the tree, we can cut off some of the overfitting. Therefore, we can get more accurate predictions by running the pruned tree on the unlabeled test set.

**10. Which members of the group worked on which parts of the assignment?**

Members of our group: Junhan Liu, Guixing Lin, Siyu Zhang

We worked together for every part of the assignment, including the code and the questions.