Rao Vinnakota
Sven Anderson
Machine Learning
HW # 3

I divided the problem into two portions - pruning the node and choosing when to prune.

Pruning the Node:
I wrote the code to prune the node without using pop(). I thought about what it meant to prune a node:

1. The node is now a leaf
2. The value of the leaf is now the major classification.

To prune the node, change the value of tree['@LEAF'] to True. Then create tree['@VALUE'] to be equal to tree['@MAJOR']. Now, to make sure that the node becomes a leaf, the algorithm iterates through the features and deletes them. Last, if it needs to, it deletes the attribute

Choosing Nodes to Prune:
I used the algorithm of reduced-error pruning as the basis for how I pruned the mush decision tree. The algorithm worked in these broad steps:

1. Create a new test tree
2. Prune the current node of the test tree
3. If that pruned test tree scores higher than the original tree
    a. prune the original tree and run the function with the new pruned tree
4. If the pruned test tree doesn't score higher:
    a. Iterate through the features of the current node calling the prune function for each node that it leads to.

For the prune_mush to work, I copied score_tree into another function return_score. In return score, the percentage correct is returned instead of printed out.

Results:
Pruning in this case was very effective. The percentage correct on the validation set rose from 80 to 86% while going from 80 to 89% on the testing set. An unwieldy tree with 1153 nodes was also lowered to a far more manageable 51 nodes. It may be better to traverse the tree a different way, rather than top-down, for a more effective attempt.