Nishant Subramani

PS2 Write-Up

1) The decision tree is represented as a dictionary where the key is the node number in a binary tree where a node X has node number k, its left child has a node number 2k, and its right child has a node number 2k+1. The value is a list of three elements for a leaf node. The first element is the outcome decision (0 or 1), the second element is a placeholder that is always None, and the third element is the number of instances in the training set that pass to this node. The value is a list of four elements for a non-leaf node. The first element is the attribute that the specific node splits on. The second element is the value on which the data is split. The third element is the number of instances in the training set that pass to this node, and the fourth element is the outcome distribution at this node, a tuple of counts of the outcome.
2) The instances in the code are read in into a list of lists, where each list in the list of lists is a specific instance, with the first list in the list of lists being the header.
3) The attribute chosen to split for each node is the attribute, when split on, that yields the largest information gain. Namely the attribute with the maximum difference in entropy between the start, before any splitting, and the post-split entropy on this specific attribute.
4) Missing examples were imputed. First an empty list was created called medians. It created an empty list for each row and iterated through it appending a value if it was not a '?'. If it was a '?', the empty list just skipped it. This medians list of lists would contain only valid numbers as values. Then we used numpy.median to compute the median values for each of the attributes regardless of type. For both numeric and nominal attributes the median was calculated. For numerical attributes, we added a slight noise term to account for the fact that if a binary variable was present and the majority were 0s, there is nothing less than 0. This way the median is 0.00001 instead of 0. Then we went back to the original data and imputed this median for all the '?' elements. (Impute_median function does this).
5) Our learning process terminates when no split exists that yields a positive information gain.
6) Without Pruning DNF Restricted to 16 Leaf Nodes: This output is 5-6 pages long on word so consult output.txt if you want to see it
7) If there is less than one injured player on the home team and more than two injured players on the opposing team, and the home team's winning percent is greater than the opposing team's winning percent, and if it has been 2 days since the previous game, and there are more than 3 starting pitchers in a game and the weather is greater than 63 degrees, and the run differential is less than 48, then the outcome will be 1.
8) Pruning was implemented by first finding a range of thresholds to try. These ranged from the fourth root of the size of the training set to the square root of the training set. Then we iterated through the thresholds by 1. If the number of instances in a node (passed as the third argument in the node) is less than the picked threshold, then the node is pruned. After that, we calculated the percent accuracy of the newly pruned tree on the validation set. This occurred at all the thresholds in the range provided earlier. We picked the tree that had the highest percent accuracy on the validation set. This tree was returned.

9) With Pruning DNF Restricted to 16 Leaf Nodes: This output is very large on word so again consult output.txt if you want to see it.
10) Unpruned Size is 9173 while the pruned is 1985, and the difference is 7188.
11) The full tree overfits to the training set a little more than the pruned set. The pruning may take some really helpful and informative nodes out however and underfit. Due to this the accuracies are similar with the pruning tree performing slightly better. The validation performance of the unpruned tree was 83.3% while the validation performance of the pruned tree was 87.36%.
12) Learning curves are included in the zip file as JPEGs. The unpruned tree had a very variable learning curve with a slight positive trend, while the pruned tree had a very positive trending learning curve.
13) Due to the pruned tree accuracy being slightly better than the unpruned on the validation set, I think that the pruned tree will work better at predicting the test set. I think our full_tree heavily overfits. Prediction test sets for both are provided in the zip file.
14) In general, a nominal attribute lends itself to fit more poorly with decision trees due to a small number of splits being possible. Continuous variables tend to perform better. However, our implementation doesn't take that into consideration and splits on the median for both nominal and numeric attributes. So in this case it does not matter very much.
15) Jeanette and I pair-programmed and pair-debugged. We alternated between working on functions. I wrote the entropy and information gain functions, came up with the pruning theory and wrote the pruning function. Jeanette wrote the make_tree function. We worked collaboratively for the rest of the code debugging each other's work.