Decision Trees and Forests

Karan Sehgal (2016CSB1080)

CSL603: Machine Learning | Lab - 1

Introduction- The goal of the assignment is to predict the sentiment of a movie review. We will be using the "Large Movie Review Dataset" from Stanford for running our experiments. The rating of the reviews are used as labels for the dataset. Any rating >=7 is a positive review while ratings <=4 are negative. Each review is represented by a bag of words representation.
We are given a list of 89k+ words and their sentiment polarities. In this assignment I have taken all words with polarity between -4.5 to -1 and 1 to 4.5. I have quantized the range of polarities into bins with size 0.01, and assigned each word, a bin, according to the word's polarity. For example, a word with polarity 1.23456 is assigned the bin with value 1.23. This bag of bins representation is used instead of the given bag of words representation.
A training set containing a random sample of 1000 observations, and a test set containing the 1000 instances was created from the train and test directories. After preprocessing, the dataset was saved as .csv files in the data directory. Each set contained equal number of positive and negative examples. A decision tree is trained according to these instances and their bag of bins representation using the ID3 algorithm. The split functions were decided on basis of whether the feature exists in the instance. Later a set of experiments were conducted on the model.

I. Preprocessing

A. Feature Building

The "Large Movie Review Dataset" contains a list of 89k+ vocabulary words alongside their sentiment polarity, as mentioned in the "Introduction", these polarities were used to cluster **18,120 words** (whose indices are stored in data/selected-features-indices.txt) into **700 bins** that are used as features for the model.

B. Sampling and Matrix Formation

Both training and test set include 1,000 samples chosen randomly from the respective directories. A validation set was also created by choosing 500 samples from the train directory which were not a part of the training set. These samples are then converted into a 1000 X 700 matrix (A), where A[i][j] contains the number of elements in instance i that belong to bin j. The labels for the samples were stored as a 1000 X 1 vector.

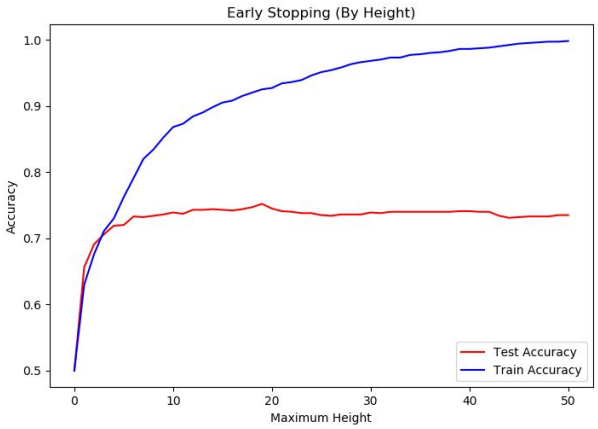
All the files were stored as .csv files in the data folder.

II. Basic Statistics

Initially the vanilla decision tree had an average height of around **60-80**, with **200-300** terminal nodes. The training accuracy stayed over **95%** for most cases while the validation and test accuracy ranged between **70-75%**. Bins with values near **-1.5 to -1** or **1 to 1.5** were most frequently used as split functions.

A. Early Stopping

Early stopping was done by two methods, one by limiting the maximum height of the tree and the other by setting some constant threshold to the change in information gain.

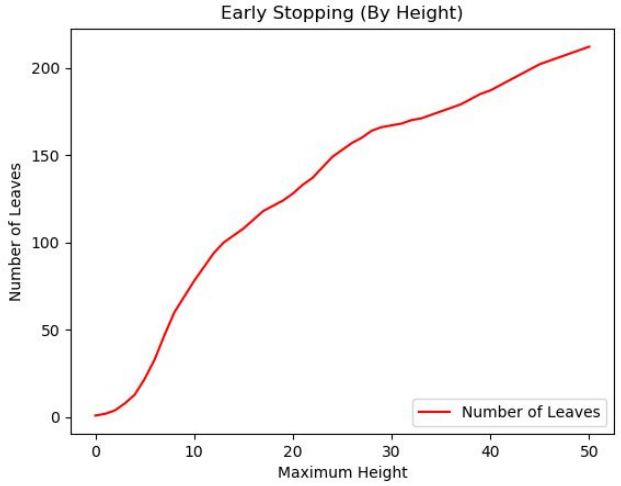


By limiting the maximum height of the tree, we saw irregular increase and decrease in the test accuracy, whereas the training increased with the increase in the height of the tree. The maximum increase in test accuracy was seen when the height of around 20-30.

The increase in training accuracy was predictable because as the height of the tree increases, it becomes more prone to overfitting. The slight increase in test data could be because finer splits (leaves having 1-2 instances) deep in the tree (due to overfitting) were removed by limiting the height, whereas it is possible there were some important splits deep in the tree which got removed as the height reduced which caused some decrease in accuracy.

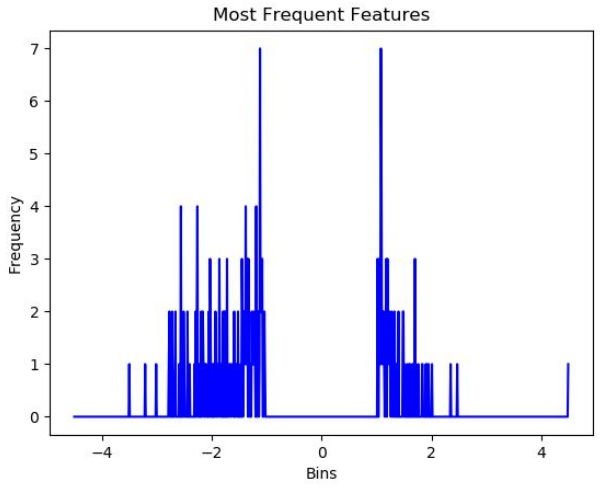
As expected, the number of terminal nodes increased with increase in height.

Early stopping was also done by having thresholds of values [0, 10^{-4}, 10^{-3}, 10^{-2}] for changing in information gain, however they didn't improve the test accuracy much.



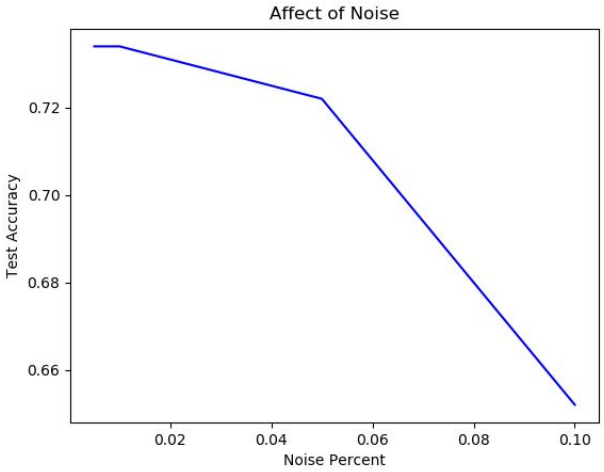
B. Most Frequent Attribute

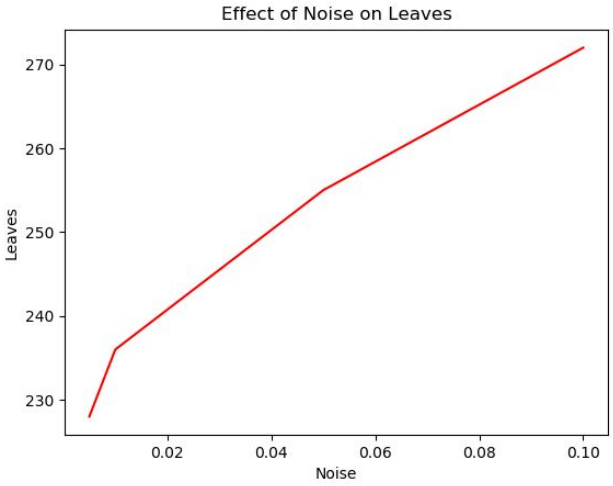
The most frequent attributes were the ones with sentiment polarity closer to zero, as they are likely to be used deeper in the tree, hence would be contained in a lot of subtrees. Bins with values -1.12, 1.08 and 1.09 were the most frequent. In the graph, since the selected polarities were between -4.5 to -1 and 1 to 4.5, the region between -1 to 1 is 0, however, if those features were also included, we'd find a bell curve as our output.



III. Addition of Noise

Noises of values [0.5, 1, 5, 10] percent were randomly added in the dataset. Labels were toggled in order to add noise. 0.5 and 1% didn't affect the test accuracy, 5% slightly decreased the test accuracy by 1-2%. We saw a drastic decrease in test accuracy by 10% noise. There were instances were the test accuracy slightly increased for 0.5 or 1% noise, the reason could be that some mildly negative reviews might have contained words having positive polarity and toggling them improved the performance of the tree slightly (since the train data had a BoW representation, a mere presence of a positive word can affect the result). Furthermore adding noise increased the number of leaves of the decision tree.

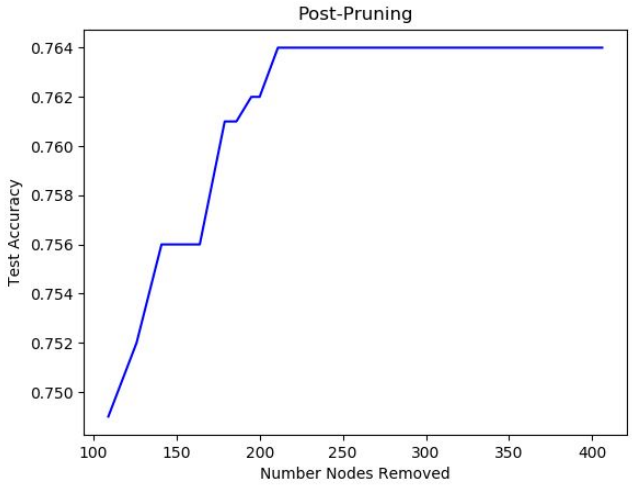




IV. Post-Pruning

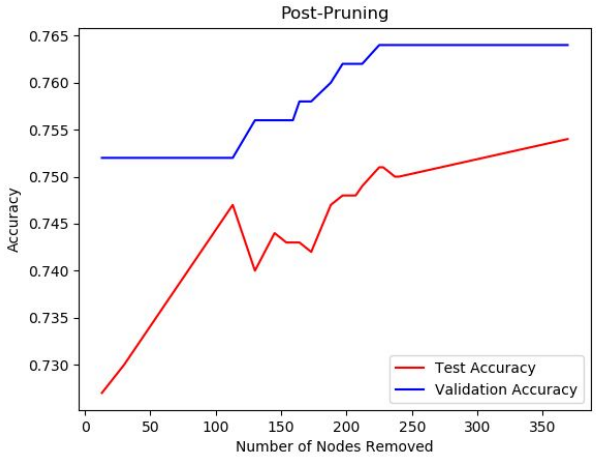
Post-pruning was done by traversing through the tree in a level-order manner. At every node, validation/test accuracy was calculated before and after the node was converted to leaf. The label on the leaf is decided by the majority i.e. if number of positive instances is more than the number of negative, the leaf is +1 and visa versa. If the accuracy after the conversion to leaf is more than the initial accuracy, the node remains a leaf and the traversal continues. The pruning ends after the traversal ends.

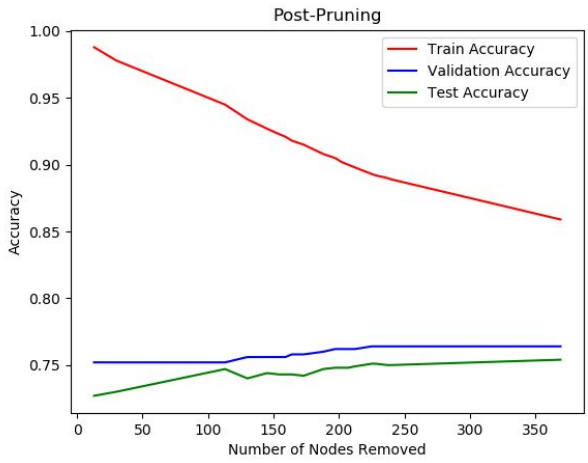
Since pruning greedily removes the subtrees having a negative effect on accuracy, removing the nodes always had a non-negative effect on the test accuracy.



The graphs below compares the effect of post-pruning on the validation set as well as the test set, when pruning was done with respect to the validation set.

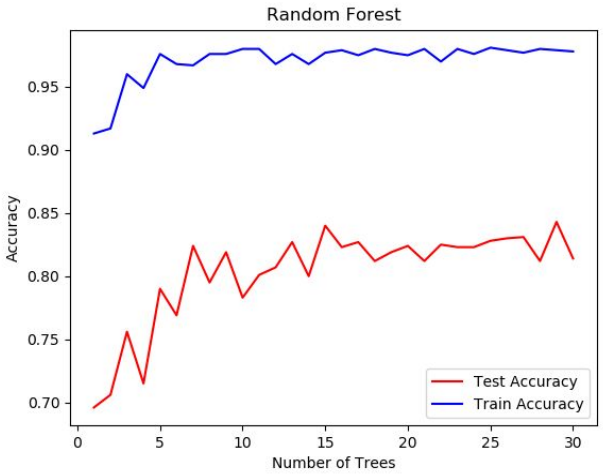
Since post-pruning is used to avoid overfitting, the accuracy of training set was decreased to 85.9%.



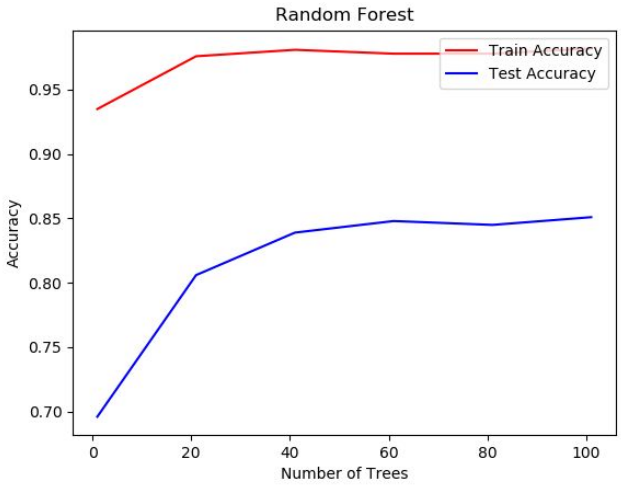


V. Random Forest

Random Forest was implemented using feature bagging. D/2 features were used at random for a tree in the forest. Since my features include D/2 positive and negative features each, having a feature bagging size of D/2 ensures that each tree would at least has one positive or negative feature. Majority voting was used for predicting the label for a test data point, if case of tie, the forest predicts positive. I also tried using $\sqrt{D}$, however it did not give good results.



By increasing the number of trees we can see that the test accuracy also increases on average.



I also noticed that odd number of trees usually gave much better result than even number of trees. It might be because there is always a clear majority with odd number of trees.