5/5/2020
CS434 Machine Learning

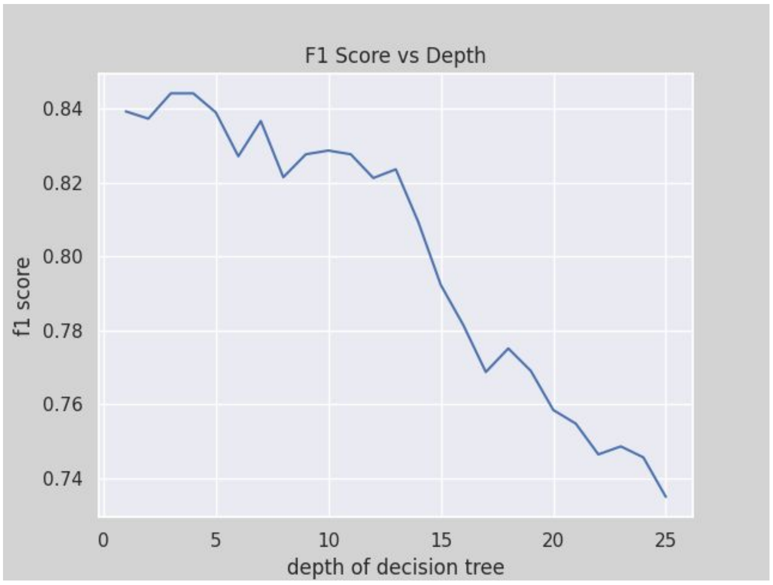
Implementation Assignment 3 Report

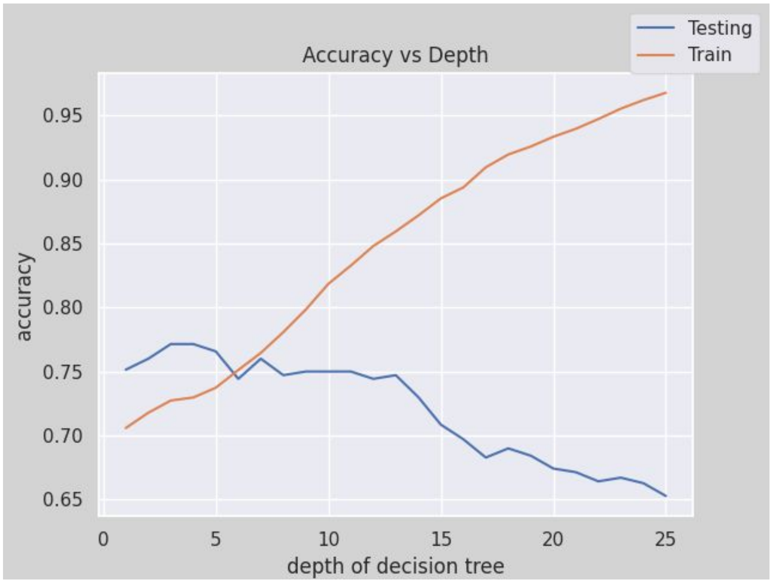
Group Members:
Lindsey Kvarfordt: kvarforl@oregonstate.edu
Aiden Nelson: nelsonai@oregonstate.edu

Workload Split: 50/50, paired programming and discussion

Part 1

Tree Depth Experiment





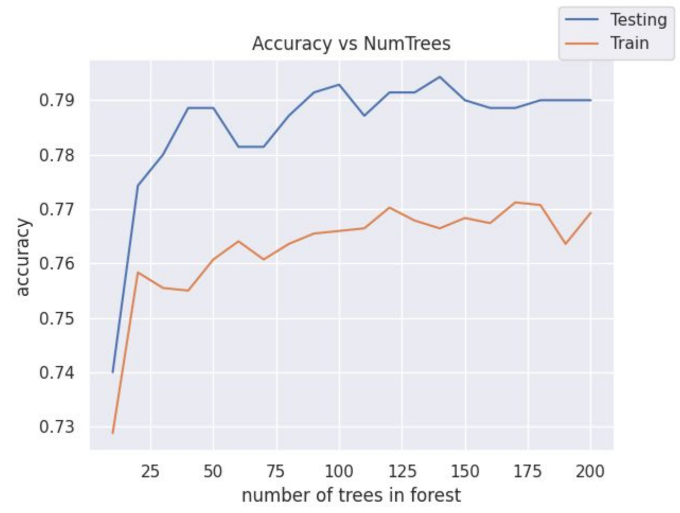
Best depth: 3
3 and 4 are the depths that give the best validation accuracy. We chose 3 to be the best depth because it results in creating the simpler model. This value was chosen based on our graph results (seen above).

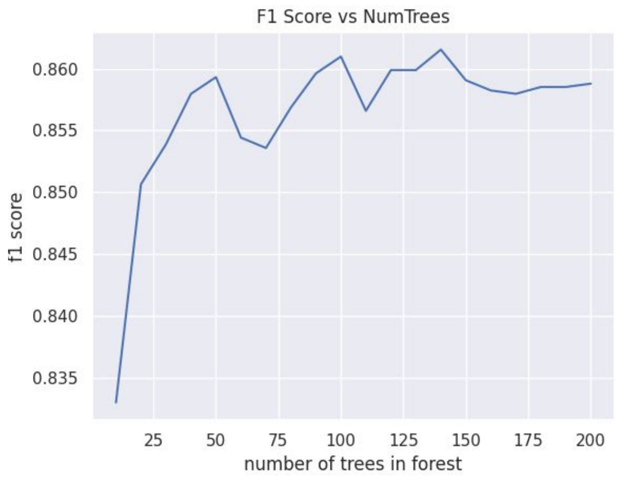
Most important feature: Feature: 8 - Description: White alone, percent, 2014
Most important feature split value: 4

The most important feature can be found by taking the first feature the data was split on. This is because the algorithm chooses the feature and value to split on based on the total benefit(gain) it will get from choosing them. Each time it will choose the optimal feature and split value that will give it the most benefit. It makes sense that the first feature and value the algorithm chooses to split upon is the most significant because it yields the greatest gain of all possible first feature-value options.

Part 2

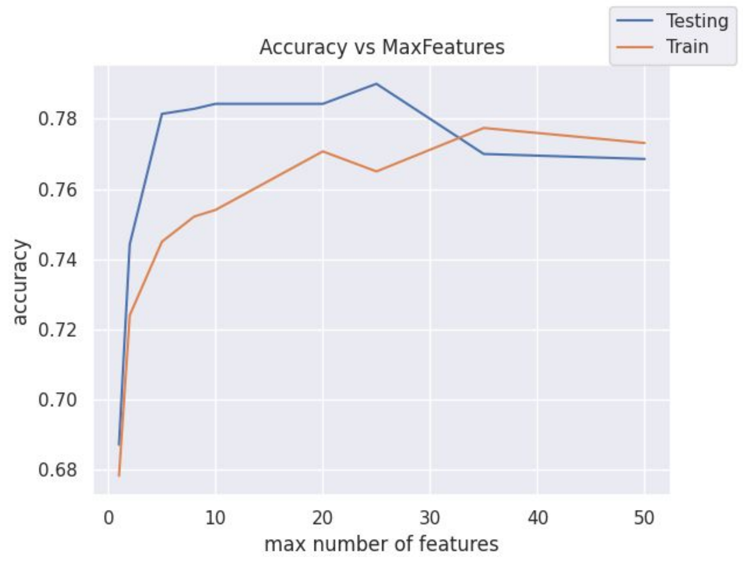
Num Trees Experiment

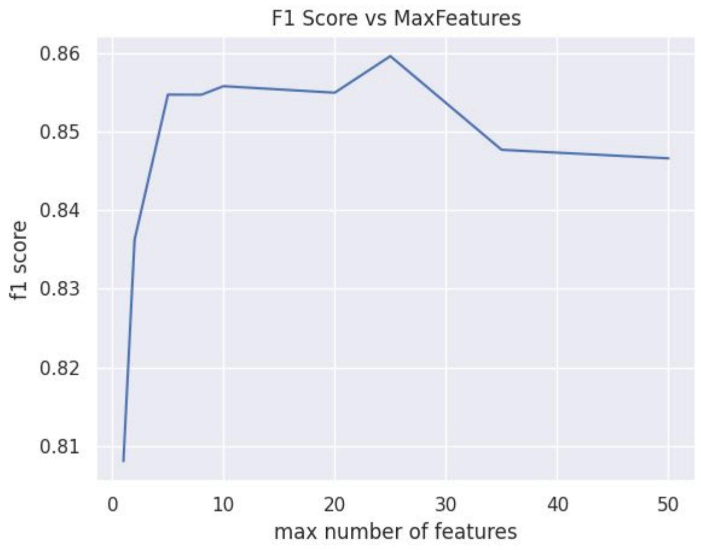




Adding more trees causes an initial general upward trend in training and testing performance. After about 50, the impact of adding more trees to the forest is kind of sporadic. The graphs above all show peak testing performance when using 130 trees. The increase in accuracy and f1 score as the number of trees used makes sense; by building an ensemble of trees to create our model, we're accounting for more variation in our training data. Each tree is built from similar data that's distributed differently, which makes the model more flexible. Eventually when enough trees are added, the increase in performance flattens out. We generated these graphs a couple of times, and the results were all similar; including up to **40 trees** increased performance, but after 40 trees, the impact of adding more was sporadic.

Max Feature Experiment





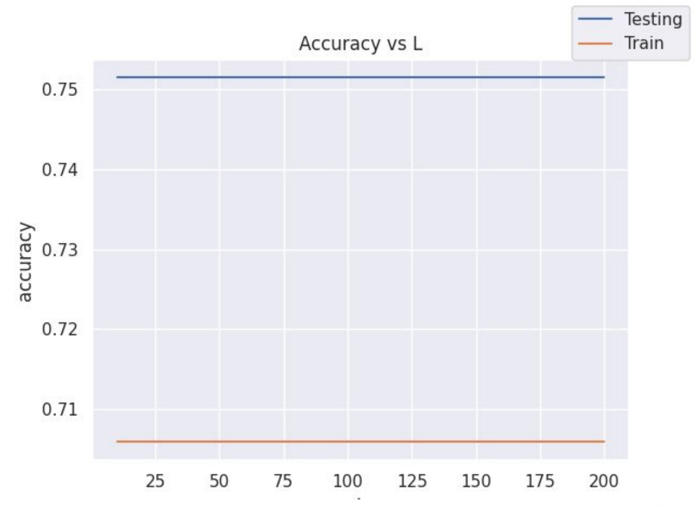
In these graphs, testing accuracy and F1 score peak at around 25 max features to be selected from per split. Again, we generated these graphs a couple of times, and they all shared the same general structure. Using 1 or 2 for max features resulted in lower performance, but using anything more than **5 max features** resulted in similar performance without a consistent trend. This makes sense because choosing less trees could be too limiting in the choice of which feature to select, but choosing more than that could lead to overfitting. In the accuracy graph, you can see that overfitting occurred after using 35 max features; after that, training accuracy surpassed testing accuracy.

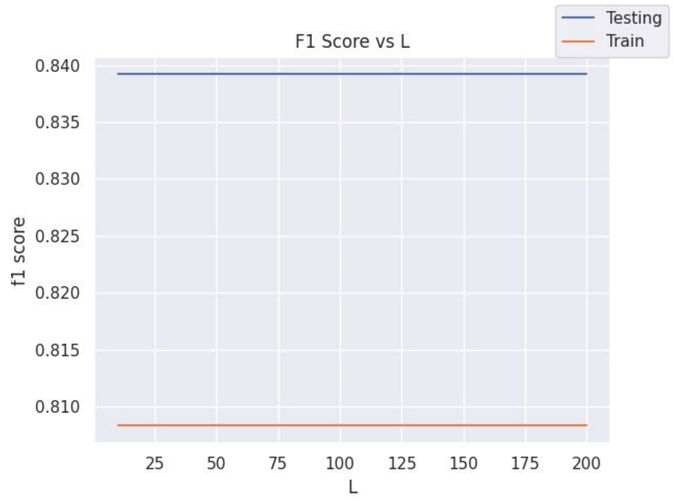
Results from maxdepth= 7, maxfeatures=25, n_trees = 130

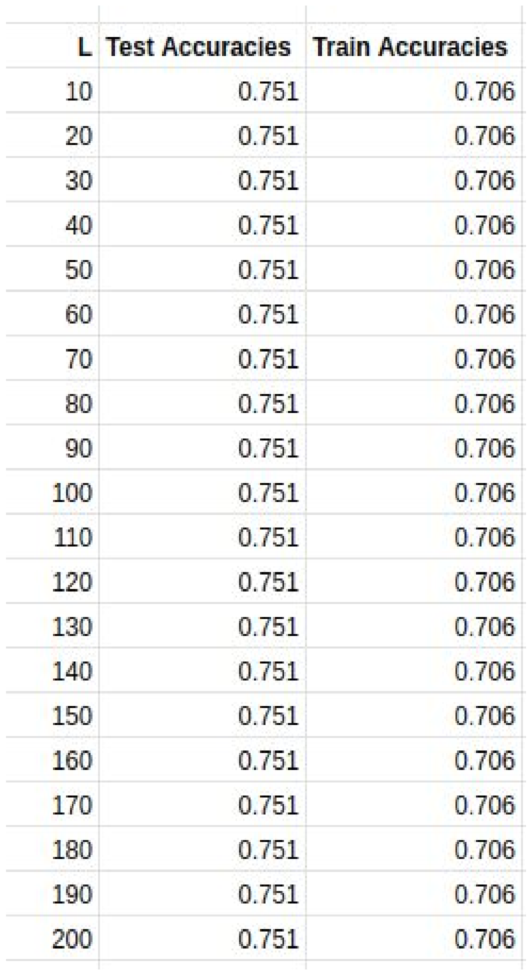
Test Accuracies	Train Accuracies	F1 Scores
0.7971	0.7798	0.8642
0.7900	0.7779	0.8591
0.7943	0.7793	0.8626
0.7900	0.7831	0.8599
0.7900	0.7798	0.8593
0.7929	0.7803	0.8610
0.7900	0.7793	0.8596
0.7943	0.7793	0.8618
0.7843	0.7774	0.8555
0.7900	0.7836	0.8593
Average, Stdev		
0.79, 0.0035	0.78, 0.0020	0.86, 0.0024

We chose to include the results of our trials with the peak values from the graphs displayed in this report instead of using the simplest model from the general trends observed. This is because these values showed ~ 1 percent improvement in all three accuracy measures for this dataset. The randomness appears to cause the accuracies and f1 scores to jitter a little bit, but cluster pretty tightly around the same point.

Part 3







L	Test Accuracies	Train Accuracies
10	0.751	0.706
20	0.751	0.706
30	0.751	0.706
40	0.751	0.706
50	0.751	0.706
60	0.751	0.706
70	0.751	0.706
80	0.751	0.706
90	0.751	0.706
100	0.751	0.706
110	0.751	0.706
120	0.751	0.706
130	0.751	0.706
140	0.751	0.706
150	0.751	0.706
160	0.751	0.706
170	0.751	0.706
180	0.751	0.706
190	0.751	0.706
200	0.751	0.706

Part 4

In an effort to help mediate and evaluate the effects of class imbalance on this data, we could plot an ROC curve to use as an additional metric of quantifying our model's performance. We could also report predictions as probabilities instead of hard class labels, and then potentially adjust the threshold of the classifications to fit the imbalanced distribution of the data. We could also undersample the majority class, which in our case is y=1, or Republican. In fact, that should probably be what we try first, as it doesn't require modification of the algorithm used.

If the dataset gets too small, which it might, we could oversample from the minority class, y=0, Democrat. I don't think we should switch to an anomaly detection framework, because our class imbalance isn't that severe.