Assignment 1, SENG 474

Quinn Webster

January 30, 2025

Introduction

The aim of this project is to classify email messages as either spam or not spam. We used a slightly modified version of the UC Irvine Spambase dataset which contains 4601 data points. The original data set had 57 numeric features, as well as its result (whether the email was or was not spam). In addition to these 57 features, our data set included 1128 more features. The original dataset can be found at https://archive.ics.uci.edu/dataset/94/spambase. Throughout this project I experimented with decision trees, random forests, boosted decision trees, and k-fold cross validation, which resulted in various models which varying levels of accuracy. All models were built from various scikit-learn libraries.

Part 1 - Separate Analysis

Decision Trees

Using the scikit-learn decision tree classifier, I first experimented with varying the percentage of data used for the training split, while keeping the testing split fixed at 0.2. I used the gini index as the split criterion. This generated some interesting results as even with a relatively small training split of 0.1, the testing error was relatively low at just under 14%. However, due to the large size of the dataset, a training size of 0.1 still involves training the model on 460 emails. This suggests that certain features may have strong alignment with spam vs non-spam classification, which would make it easier for the model to learn with limited data. As the training size increased, we see a general decrease in testing error although it was not a perfect linear decrease, which could be caused by complexity in the dataset. Throughout varying training set sizes, the training error stays almost constantly at 0, with a slight increase while having larger training sizes, this is as expected due to the greater the training size, the more difficult it is to perfectly classify each datapoint with more data. We then repeated this experiment while using entropy as the split criterion, and obtained similar results, with a slightly higher testing error at the beginning of the experiment. This indicates that, in this case, the choice of splitting criterion has minimal impact on model performance. For all other experiments regarding decision trees, I kept the training split size constant at 0.8 and the testing size at 0.2. The results from this experiment are shown in Fig 1 and Fig 2

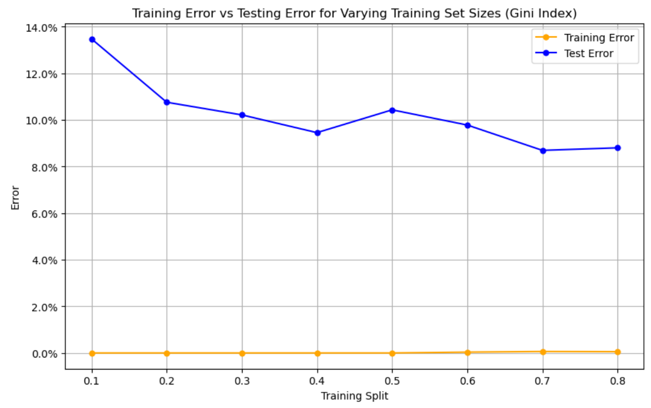

Figure 1

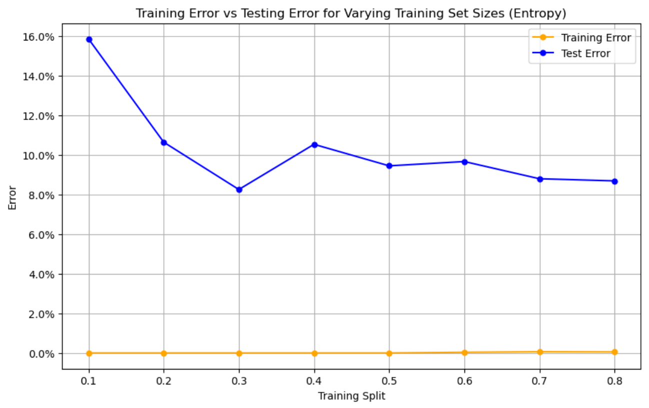

Figure 2

I then proceeded to experiment with varying maximum depths of the decision tree. Using the Gini index as the split criterion, I built 29 decision trees with depths ranging from 1 to 30 (Fig 3, Fig 4). As expected, with a maximum depth of 1, both the training error and testing error were quite high (around 17.5%). However, when the maximum depth was set to 4, the training error dropped to 7.5% and the testing error to about 9%. This was interesting, as even with a relatively low maximum depth, the error is quite low. As the depth increased from 4 to 13, the testing error gradually decreased to 7%. As the depth increased from 13 to 23, there is a slow incline back up to around 9%. This demonstrates how the model begins to slightly overfit, so a bigger tree actually results in a less accurate model. When the maximum depth was between 24 and 29, the testing error plateaued at 9%, showing that the tree depth did not exceed 24. The training error continues to decrease with increasing depth until it leveled off at a value of 24 (same as the testing error max depth). When repeating the experiment using entropy as the split criterion, we obtained very similar results. The testing error was minimized at a depth of 13, and the tree reached its maximum depth at 24.

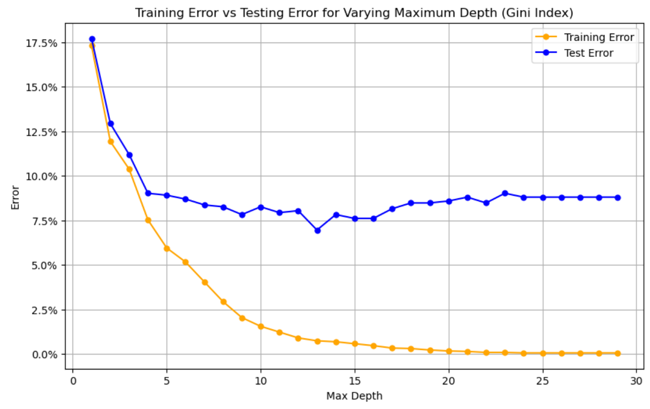

Figure 3

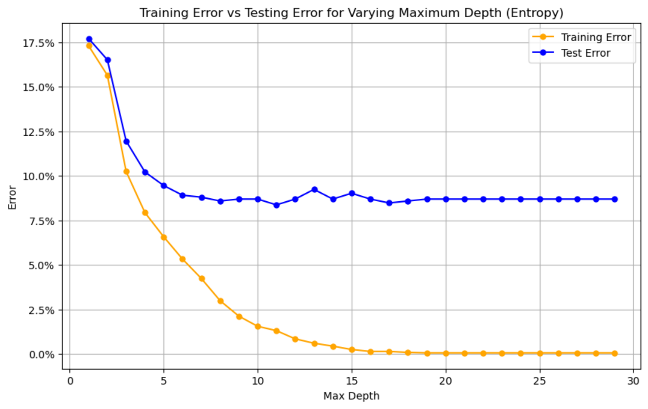

Figure 4

To explore limiting overfitting of the tree, I next varied the minimum samples per split parameter, while using the Gini index as a parameter. This can be an important parameter to test with as it can reduce overfitting a model; as it controls how many samples are needed to split a node, when given a higher value, it doesn't allow for smaller splits. However, if set too high, it can lead to underfitting. With a value of 2, the training error is 0%, indicating that as long as there are more then 2 samples, the tree can split node into child nodes, classifying each datapoint as either spam or not-spam very well. While this does results in zero training error, it may lead to an overfit model, which is demonstrated in Fig 5 with a testing error of 8.9%. When we increase the value of minimum samples per split to 8, we receive a higher training error of 2.8%; however, the testing error falls to just under 8%. This demonstrates the importance of balancing training and testing error. As we increase the value past 16, both the training error and testing error increase. This happens because with fewer splits, many datapoints must be grouped together. When repeating this experiment with entropy (Fig 6) as the splitting criterion, we received similar results.

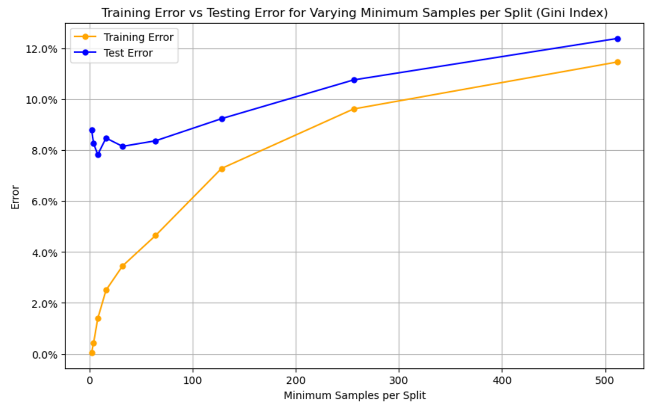

Figure 5

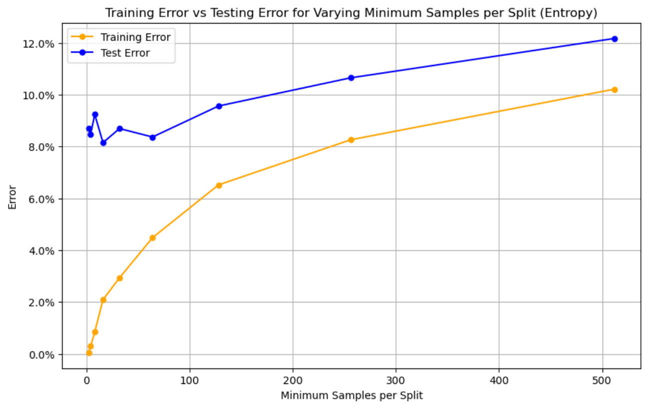

Figure 6

I implemented reduced error pruning to the decision tree to understand its effect on reducing overfitting, while retaining the most important nodes of the tree. The pruning algorithm uses a post order traversal to iterate over each node, and check to see if the model would gain accuracy if it removed its two child nodes. If so, it would do so, and if not the node would keep its children, and the algorithm would move on to the next node.

Fig 7 shows the decision tree built with the Gini index as the split criterion prior to pruning. This tree is quite complex, with 341 nodes and a testing error of 8.3%. Fig 8 shows the tree after pruning, which is much simpler with 266 less nodes, and with a reduced error of 7.1%. This is interesting see as it shows that having a smaller, pruned tree can actually result in a better trained model by removing the unnecessary complexity.

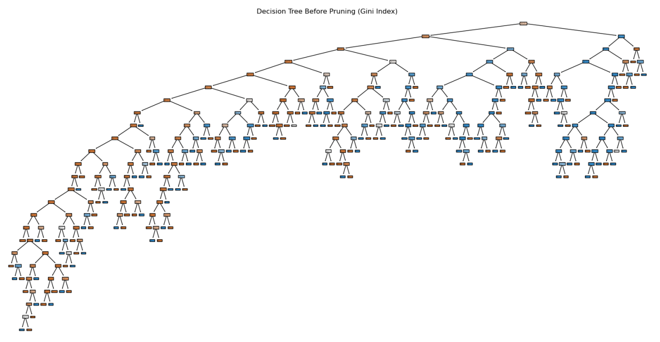

Figure 7

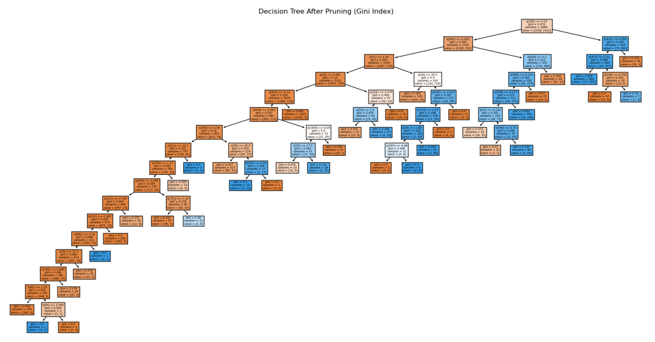

Figure 8

Although the pruned tree is significantly less complex than the original, I wanted to experiment with a pruning model that would be more likely to remove a node, resulting in a simpler tree. I modified the reduced error pruning algorithm to include a slight threshold, so that we would still prune a nodes children even if the error of the resulting tree would be slightly increased, as long as it was under a certain threshold. This could be used in case you are content with the trade off of a much smaller tree for a greater level of error. I ran this experiment with multiple threshold as shown in Fig 9 and Fig 10.

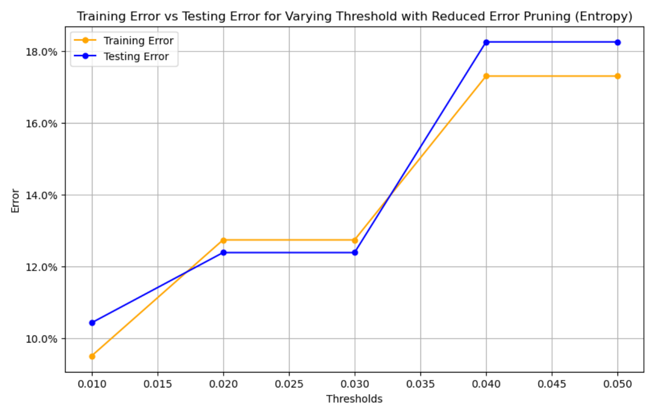

Figure 9

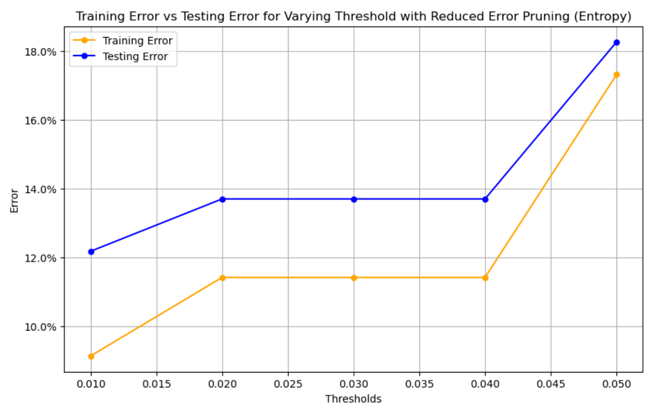

Figure 10

Although adding a threshold allowed for much simpler trees (less nodes), the increased error is dramatic. I obtained similar results while using both the Gini index and entropy as the split criterion, reaching error levels around 18% when working with the largest threshold value. I would not recommend modifying ones pruning algorithm like this, unless you used very small threshold values (less then 0.01) or did not care about the high increase in training error and only cared about a simplified tree.

Random Forests

For the next set of models, I used the Random Forest Classifier from Scikit Learn. Again, as an initial experiment to obtain a general overview of how random forests would perform on this training set, I examined how varying the training data size while keeping the testing data size constant affected performance of the model. I chose a constant testing set size for better comparability of results, rather than making it the complement of the training set size. I initially used the Gini index (Fig 11). The results matched my expectations and were similar to those of the decision tree experiment. The testing error continuously decreased as the size of the training data increased, which is a result of having more data to accurately train my model. While performing this experiment I kept the number of estimators (trees) at a constant value of 100, which is a reasonable value to give the random forests enough training points, while likely not risking overfitting. I repeated the experiment using entropy as the split criterion for the decision trees (Fig 12) and received similar results, highlighting that criterion does not significantly affect the accuracy of random forests.

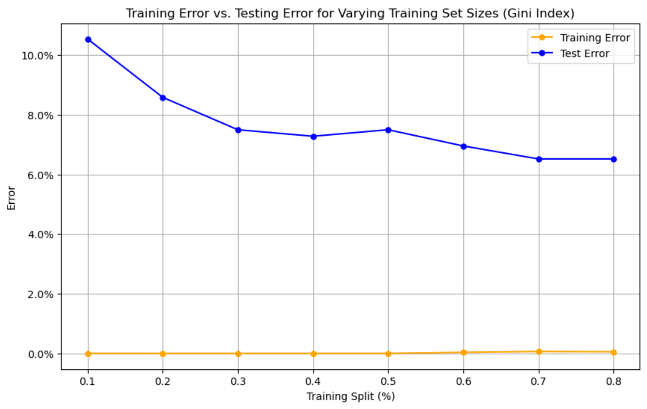

Figure 11

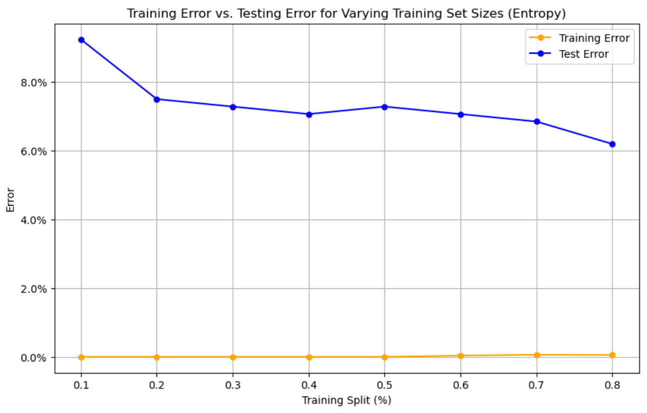

Figure 12

Next I modified the number of trees that I would train the random forest with, working with a wide range of values, from 1 to 200. I ran this experiement using both the Gini index and entropy as the split criterion (Fig 13, 14). The results of this experiment were interesting, between 20 and 200 trees, there was only a very slight decrease (0.25%) in the testing error. Even though we ran the model with 10 times as many trees, it hardly had an effect on the testing error. While working with entropy as the split criterion, we received somewhat similar results, except that we obtained a new minimum value of testing error (4.5%) when we ran the model with 50 trees. Although this was only 0.2% lower error then when we ran the model with 15 trees. This demonstrates how in the case of this data set, the number of trees (with a minimum of 15) does not have a significant affect on the testing error. These results demonstrate that the number of trees (above a reasonable minimum) does not have a significant impact on testing error. Additionally, the split criterion seemed to have little effect on developing a superior model.

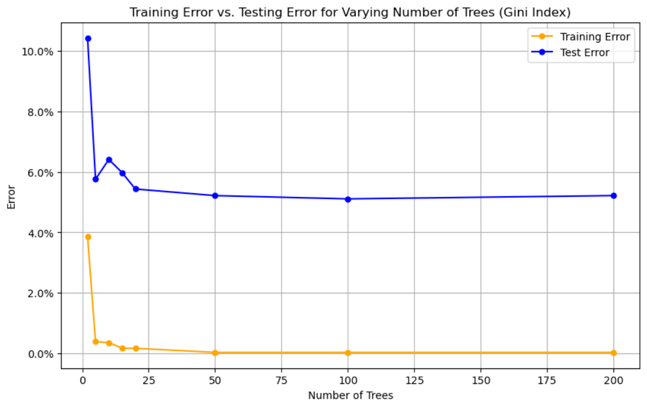

Figure 13

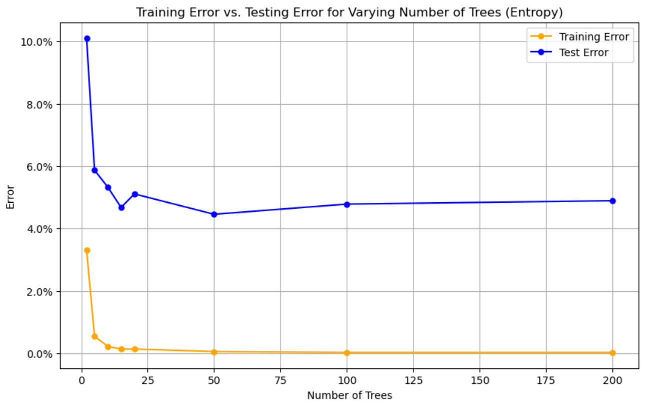

Figure 14

I proceeded to run an experiment varying the number of random features of the random forest. A lower value for random features will increase the randomness between trees, resulting in more independent trees, which may perform better on unseen testing data. However, each individual tree will be exposed to less data, which could reduce the accuracy of it. On the other hand, a higher number of random features will cause trees to become more correlated with each other since they share so much data used for splits, which may lead to overfitting. My results showed minimal variance on the testing accuracy, maintaining values between 5.6 and 6.5 while using the Gini index (Fig 15) as the split criterion, while using entropy (Fig 16) resulted in a range of 5.8-6.2. This suggests that for this dataset, the number of random features is not as significant in building a model. However, it is still important to note the balance between tree independence and correlation to prevent underfitting and overfitting a model. In future experiments with different datasets, I will still consider running similar experiments to evaluate usefulness.

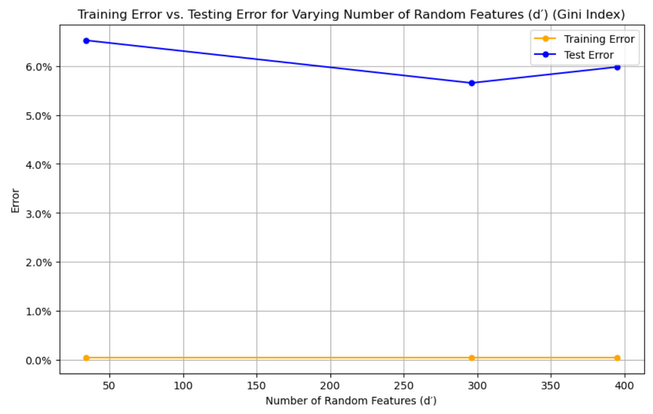

Figure 15

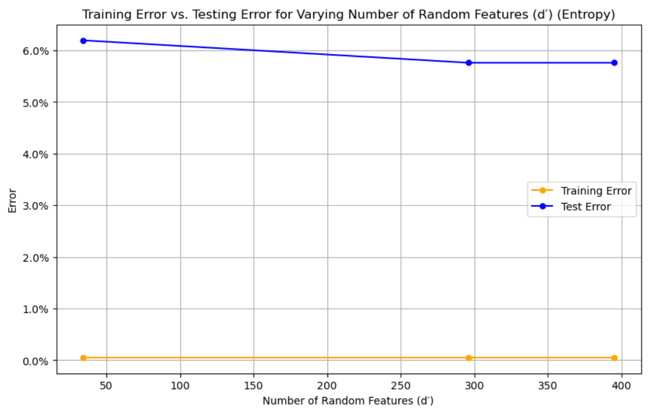

Figure 16

Boosted Decision Trees

While varying the training set sizes in a range from 0.1 to 0.8, and keeping the test split size constant at 0.2, we obtained consistent results with previous experiments involving varying the training split size. The testing error generally decreased as the training split size increased. Due to the fact that having more data to train a model, allows it to better adapt to unseen testing data. However, as opposed to previous results, the training error increase quite a bit as the training split size grew. This is likely due to the sequential and weighted attributes of AdaBoost. As the training set grows, there are more challenging cases for the model to correctly predict, and since the weaker learner is so shallow, it struggles to predict it perfectly. This makes it more difficult for the model to perfectly predict the training error; however, it can help reduce the testing error. (Fig 17, 18)

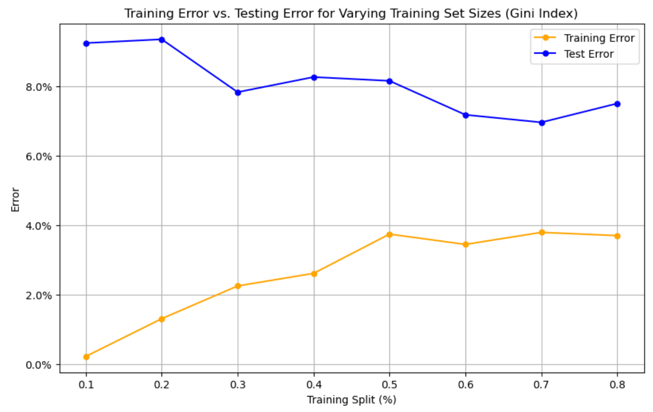

Figure 17

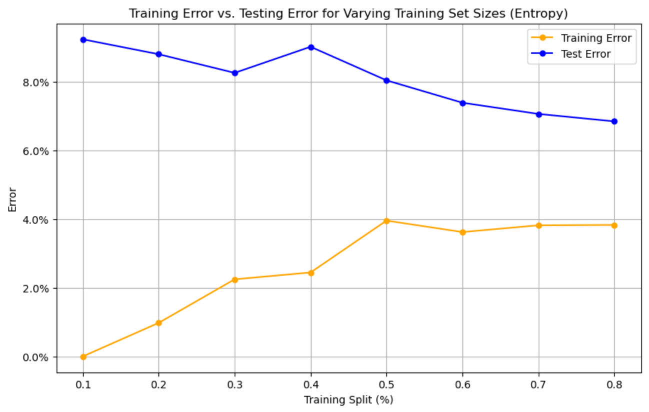

Figure 18

When we increase the maximum depth of the weak learner, the deeper trees are more complex and are more capable of fitting the more challenging datapoints. This will result in a reduced training error, as the weak learners can now better fit all of the data points. However, there can also be a plateau in the results of the testing error. Since AdaBoost is supposed to rely on very weak learners, when they are made more complex, each weak learner individual prediction is more accurate, but that's not really going to help the AdaBoost model, as it will now be more focused on fitting to the data instead of generalization. But slightly increasing the maximum depth to 2, may benefit the process, as now each weak learner will be able to split data on two features instead of just one. Although this will still be quite a weak learner, it may be able to classify data points with a slightly improved point. These two points can be seen in Fig 19 and Fig 20, where the training data decreases to 0 when the maximum depth increases, and the testing data remains relatively flat.

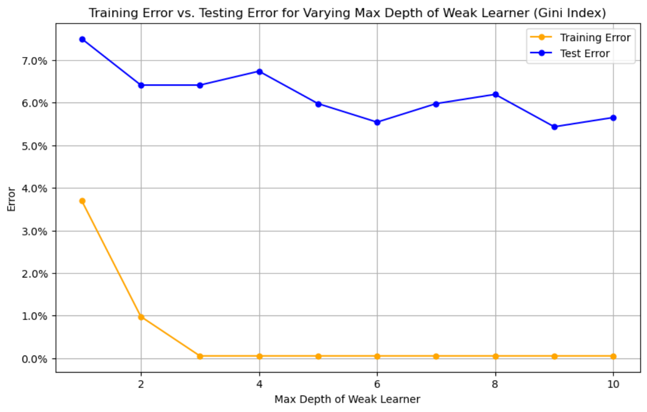

Figure 19

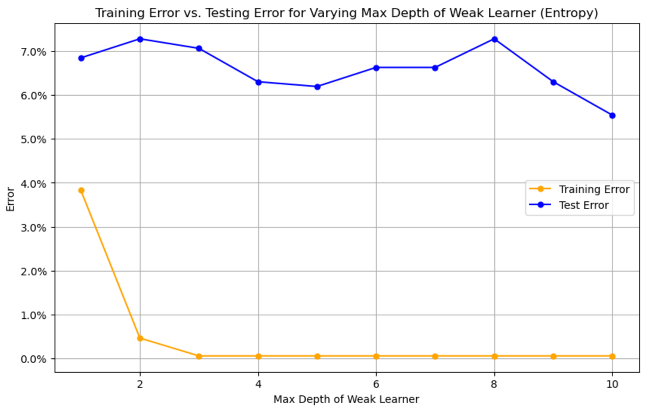

Figure 20

When varying the number of weak learners, we can see that between 10-50 weak learners, we get a steady decrease in testing error. However, between 50 and 400 weak learners we keep a relatively constant value of error, with a variance less then 0.5%. We achieved very similar results when using entropy as our split criterion, which is as expected, because since the max depth of each weak learner is 1, the first split generated by the Gini index and entropy are often quite similar, leading to similar results. (Fig 21, 22)

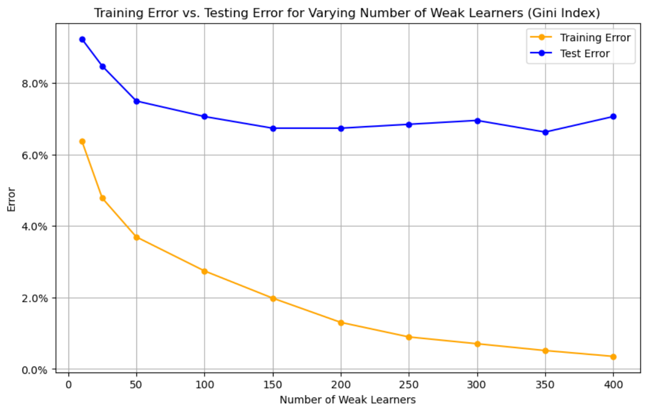

Figure 21

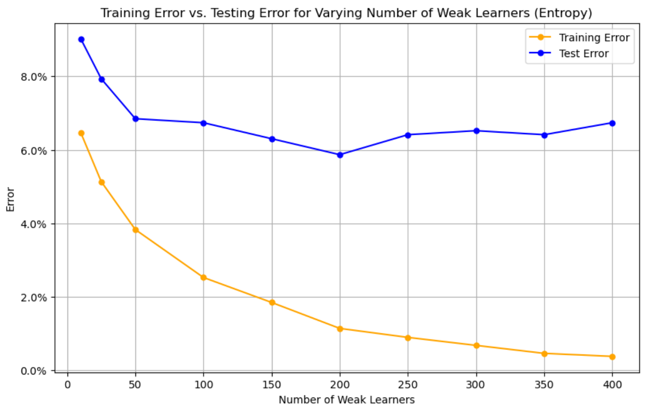

Figure 22

Part 2

For this experiment I used k-fold cross validation to determine the optimal number of trees in a random forest that minimizes error. I kept the hyperparameters max_features='sqrt' and criterion_split = 'gini' as the static hyperparameters. Based off of my results from varying the number of trees in a random forest in section 1 (Fig 13, 14), I set the values in the ensemble to range from 50 to 300. The results (Fig 23) showed that there was not a large difference in average error on the training data (less then 0.12%); however, the two values that achieved the lowest training error were 175 and 200. These are reasonable values, as 175-200 are standard numbers of trees to use in a random forest. After training a new random forest model with 175 trees, I received a training error of 5.11%. This is a very low error, and it demonstrates the effectiveness of k-fold cross validation. I was able to find a value for number of estimators that resulted in a great testing performance. The results also alligned well with my results from section 1 (Random Forests).

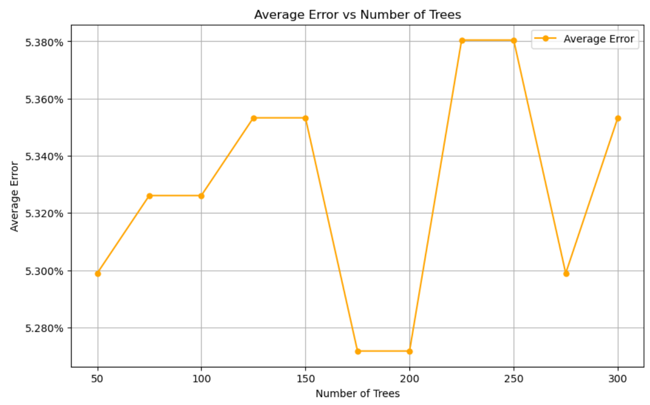

Figure 23

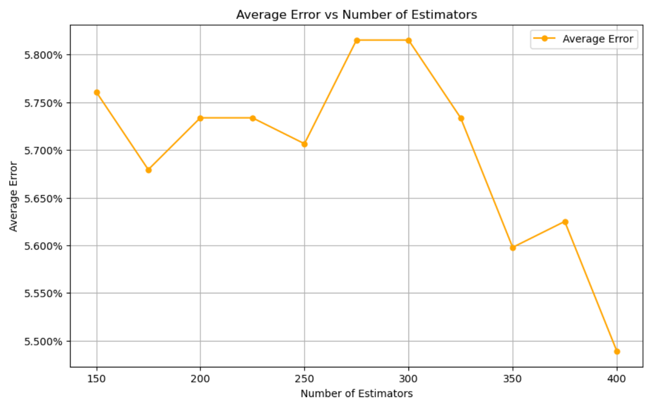

Figure 24

I then used k-fold cross validation to tune the number of estimators that I was using for my AdaBoost model. Based on my results from section 1 (Fig 17,18), I expected a value between 150 and 400 estimators that would result in the lowest error. After using k = 5, Fig 24 shows the average error across the k-fold cross validation for each value of number of estimators. Number of estimators having a value of 400 had the lowest average training error. With this tuned value, I then trained a new AdaBoost model and obtained a testing error of 5.1%. This was an even lower value then the tuned random forest model. With 400 estimators, the model likely found an effective balance between correctly predicting misclassifications while also avoiding overfitting. This low error demonstrates the effectiveness of AdaBoost on this particular dataset.

Having trained decision trees, random forests, and AdaBoosted trees, it was interesting to explore the results obtained from all of these experiments. It was also interesting to see how tweaking certain parameters could results in underfitting or overfitting a model, or in the case of some experiments, not always make as big of a difference as potentially expected.