SENG 474 A02: Assignment 1

Binary Classification Experiments: *Decision Trees, Random Forests, and Neural Networks*

Sean McAuliffe, V00913346

February 4th, 2023

Contents

Introduction

1 Background

This report summarizes the results of a series of experiments conducted to measure and compare the performance of three different methods of binary classification: decision trees, random forests, and neural networks. For each of these approaches a number of models were trained on aa provided dataset containing census information of american adults, with the goal of predicting whether or not an individual makes more than $50,000 a year.

Each experiment consisted of splitting the dataset into a training set and a test set, and varying a single training hyperparameter of interest while measuring the resulting performance of the model. Each experiment produced a graph of the model training and test accuracy (or error) as a function of the hyperparameter.

Some of the experiments tested different combinations of parameters such as optimizationm algorithms, and so did not produce a chart of accuracy as a function of the hyperparameter because the possible parameter values were categorical, not numerical. In these cases the results are presented in a table.

2 Environment & Implementations

The three classifier methods were not implemented from scratch for the purposes of this assignment. Instead, the **scikit-learn** library was used. The experiments were conudcted in Jupyter Notebooks running on a Python 3.10.6 kernel (not the default ipython kernel). The notebooks have been provided with this report. An associated README file constains intructions for running the notebooks in order to replicate the results.

Decision Trees

3 DT Hyperparameters

Decision trees are a supervised learning method which can be used for classification and regression. A tree consists of a number of nodes which represent a question about some feature of the data, here the remaining examples are split into children nodes according to some threshold of the feature value. At leaf nodes, the model learns to predict a classification label by inspecting the labels of the provided training examples which have ended up at this node and using a majority vote. Future testing examples which have feature values which lead to this leaf node will take on this predicted classification.

Scikit-learn provides a **DecisionTreeClassifier** class which can be used to perform the binary classification experiment. The following hyperparameters are of interest.

- **criterion**: The function used to measure the quality of a split.
- **max_depth**: The maximum depth of the tree.
- **ccp_alpha**: Complexity parameter used for Minimal Cost Complexity pruning.

4 Decision Tree Classifier Experiments

4.1 Maximum Tree Depth

Experiment 1: The data was split according to a 80% - 20% training-testing split. The maximum depth of the DTC was varied from 2-100. For each value of max_depth a new model was trained on the training data and scored against the testing data. The results are shown in Figure **??**.

The results of experiment 1 (without pruning) show a clear pattern of overfitting. The training error decreases to near zero as the tree depth increases. While the testing error initially decreases, but then quickly begins to rise

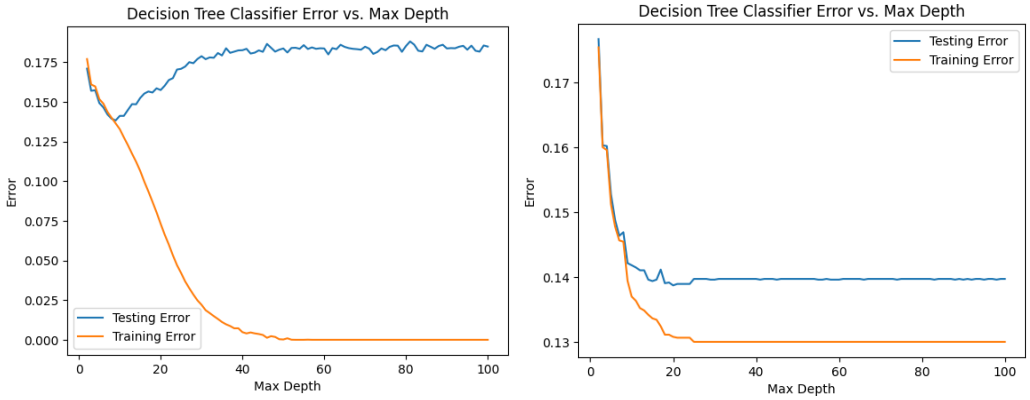

Figure 1: DTC Error Rate vs. Maximum Tree Depth (Top: w/o pruning, Bottom: w/ pruning)

and plateaus at a value of approximately 0.175. This is seemingly prevented with the inclusion of post-pruning.

Minimal cost complexity post pruning is used here to prevent over- fitting. This method of post pruning is included in the **DecisionTreeClassifier** class in **scikit-learn** and is enabled by setting the **ccp_alpha** hyperparameter to a non-zero value (the default is zero). The optimal value of **ccp_alpha** was found in a later experiment and was then applied here.

The application of post pruning reduces the final testing error to approximately 0.14; a fairly significant improvement. The testing error plateaus after a tree depth of approximately 20, and further increasing the depth does not increase the testing error; suggesting that this method has successfully prevented the model from overfitting.

4.2 Complexity Parameter

Experiment 2: The data was split according to a 80% - 20% training-testing split. The complexity parameter **ccp_alpha** of the DTC was varied from 0-0.005. For each value of **ccp_alpha** a new model was trained on the training data and scored on the testing data. The results shown in Figure **??**.

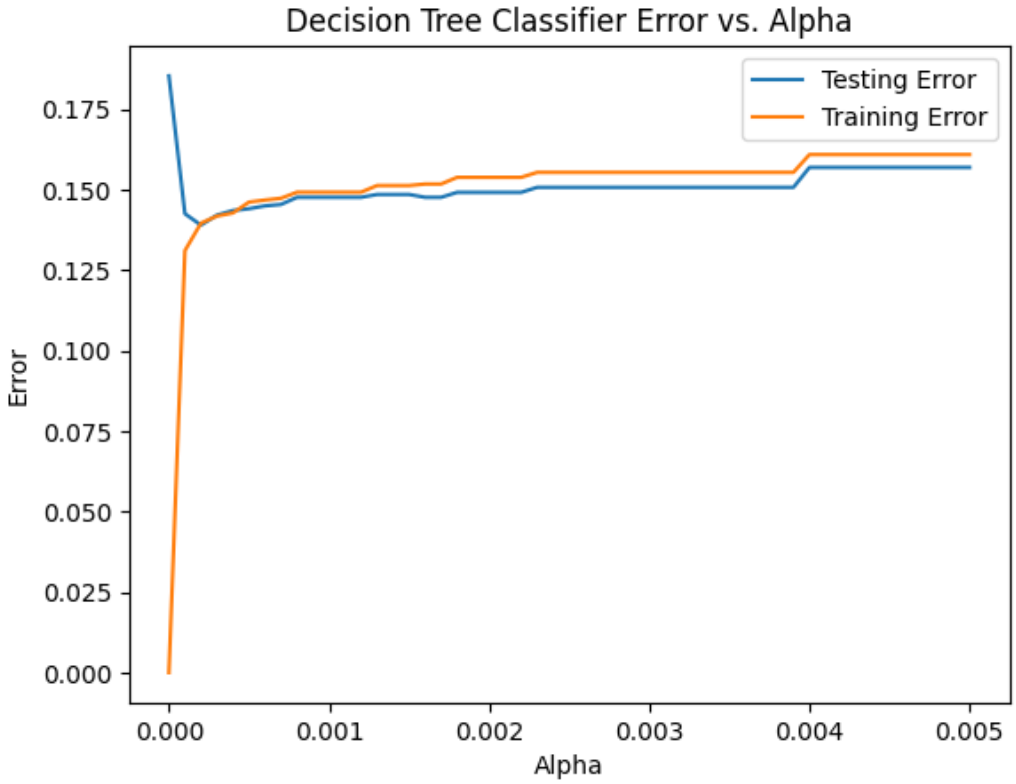

Figure 2: DTC Error Rate vs. Complexity Parameter (Top: w/o pruning, Bottom: w/ pruning)

The results show that with no pruning (**ccp_alpha** = 0.0) the model overfits, with the training error near zero, and a relatively high testing error. As the value of **ccp_alpha** increases, the model training and testing error converge on a value of approximately 0.15. The optimal value is found to be approximately 0.0002. This value has been applied to the other experiments in retrospect.

The depth of the tree resulting from trainig without prunning tends to be around 54. The depth of the tree resulting from training with pruning tends to be around 12. Reducing the total number of nodes in the tree from 10831 to 107. This is a significant improvement in the performance of the model.

4.3 Proportion of Data in Training Set

Experiment 3: The proportion of the data used for training was varied from 10% to 90%. For each proportion a new DTC model was trained on the training set and scored on the testing set. In the results below the graph plots the accuracy of the model, not its error. The results are shown in Figure **??**.

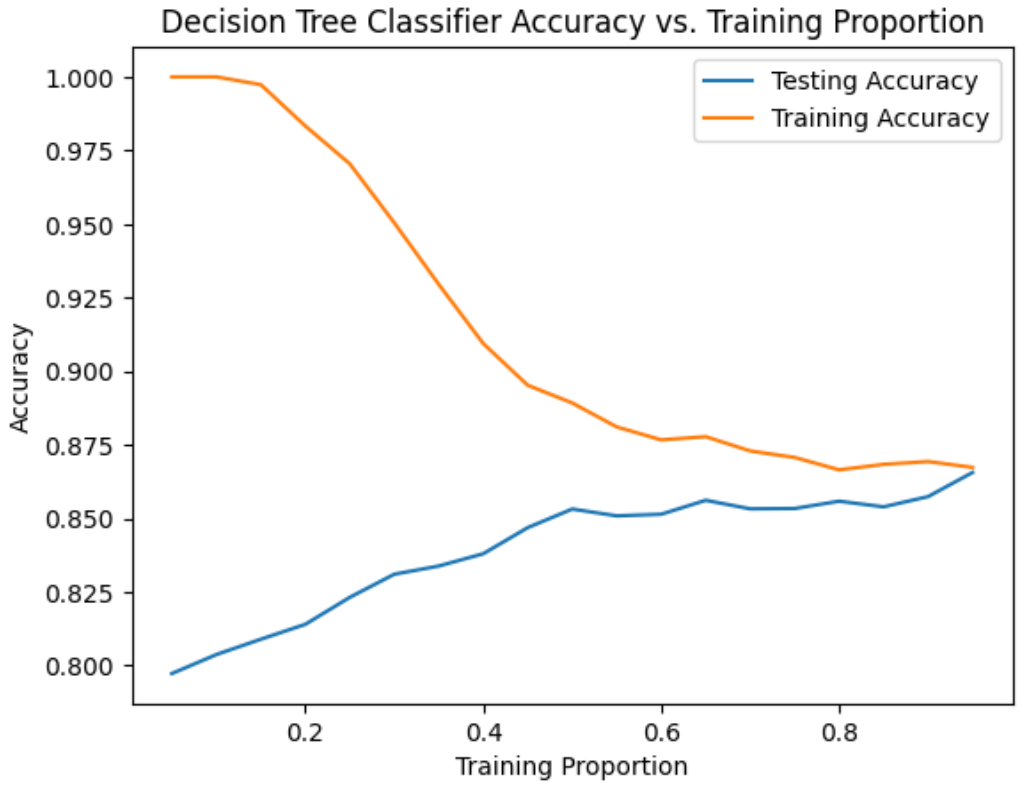

Figure 3: DTC Accuracy vs. Proportion of Data in Training Set (w/ Pruning)

As an extension of this experiment, the model accuracy was again measured with respect to the proportion of data in the training set, but this time no post-pruning was used. The results (shown in figure **??**) appear to show that the DTC model overfits when allowed to go to an arbitrary depth, and that the degree of overfitting is not influenced much by the training testing set

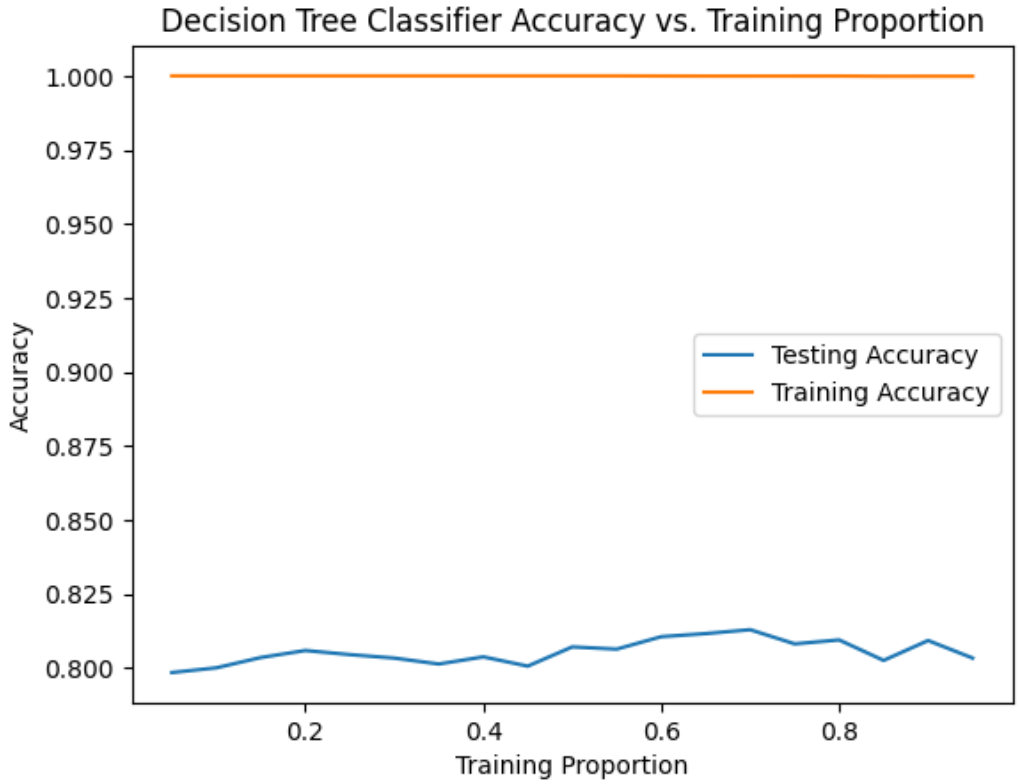

Figure 4: DTC Accuracy vs. Proportion of Data in Training Set (No Pruning)

proportions. I had expected larger training sets to be harder to overfit-to, but this does not appear in the results.

4.4 Criterion

Experiment 4: The criterion used for splitting was varied between **gini**, **log loss**, and **entropy**. For each criterion a new DTC model was trained on the training set and scored on the testing set. The results are shown below. The results appear to imply that entropy and log-loss crieterion are more sensitive to overfitting, and that the best testing accuracy can be achieved with the gini criterion.

```
Training Accuracy:
```

```
    Gini:       0.86944
    Entropy:    0.91187
    Log Loss:   0.91149

Testing Accuracy:
    Gini:       0.85682
    Entropy:    0.84002
    Log Loss:   0.83957
```

Random Forests

5 RF Hyperparameters

Random Forests are an ensemble classification method which consrtuct many Decision Trees, each one trained on a bootstrap sample (a subset of the full training set). Each tree is trained on a random subset of the features. The final prediction of the Random Forest is made by averaging the predictions of each of its Decision Trees. This method can be very effective at certain kinds of problems.

Scikit-learn provides a **RandomForestClassifier** class which is used in this section. The hyperparameters of the Random Forest Classifier which are of interest in the following experiments are:

- **n_estimators**: The number of trees in the forest.
- **max_depth**: The maximum depth of each tree.
- **max_features**: The max. number of features to consider for a split.
- **max_samples**: The max. number of samples drawn from the training set to train a single estimator.
- **min_samples_split**: The min. number of samples required to split an internal node.

6 Random Forest Classifier Experiments

6.1 Forest Size

Experiment 5: The number of trees in the forest was varied from 2 to 200 in steps of 5. For each value of **n_estimators** a new Random Forest model was trained on the training set and scored on the testing set. This experiment used an 80% - 20% training-testing split. The results are shown in Figures **??**, **??**, and **??**.

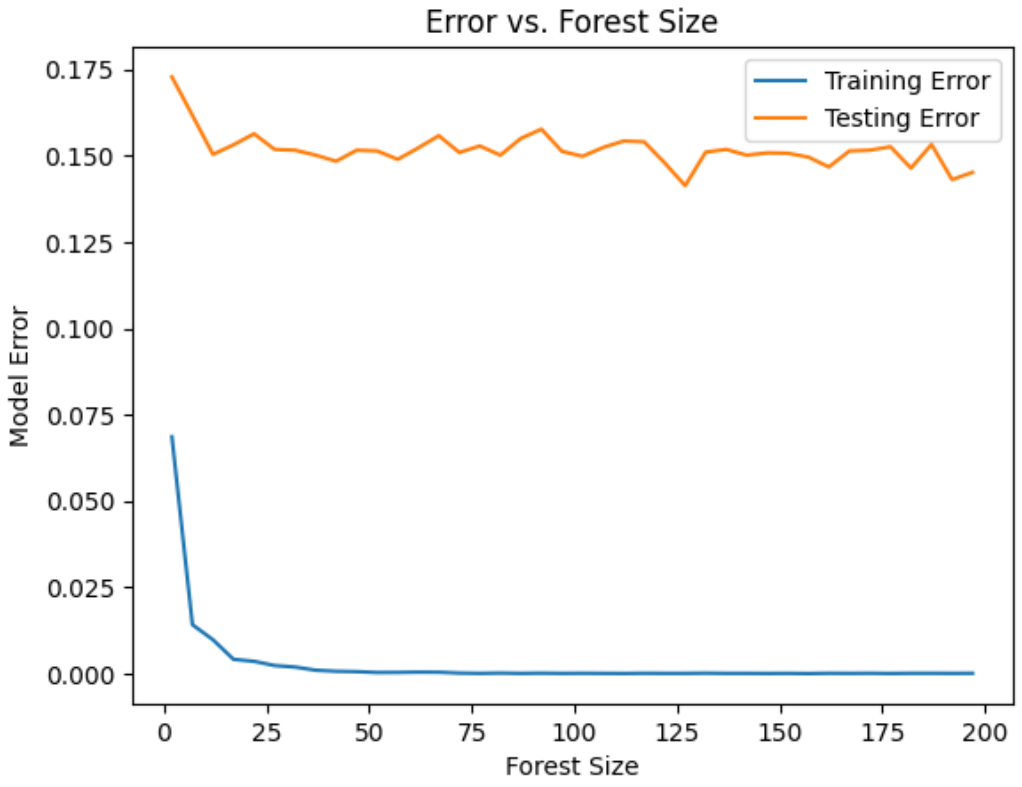

Figure 5: RF Error vs. Forest Size (dynamic set)

The results shown in Figures **??** and **??** show the accuracy of models each of which was trained and scored on a differntly shuffled dataset (dynamic set) to maximize the randomness between iterations of **n_estimators**. The

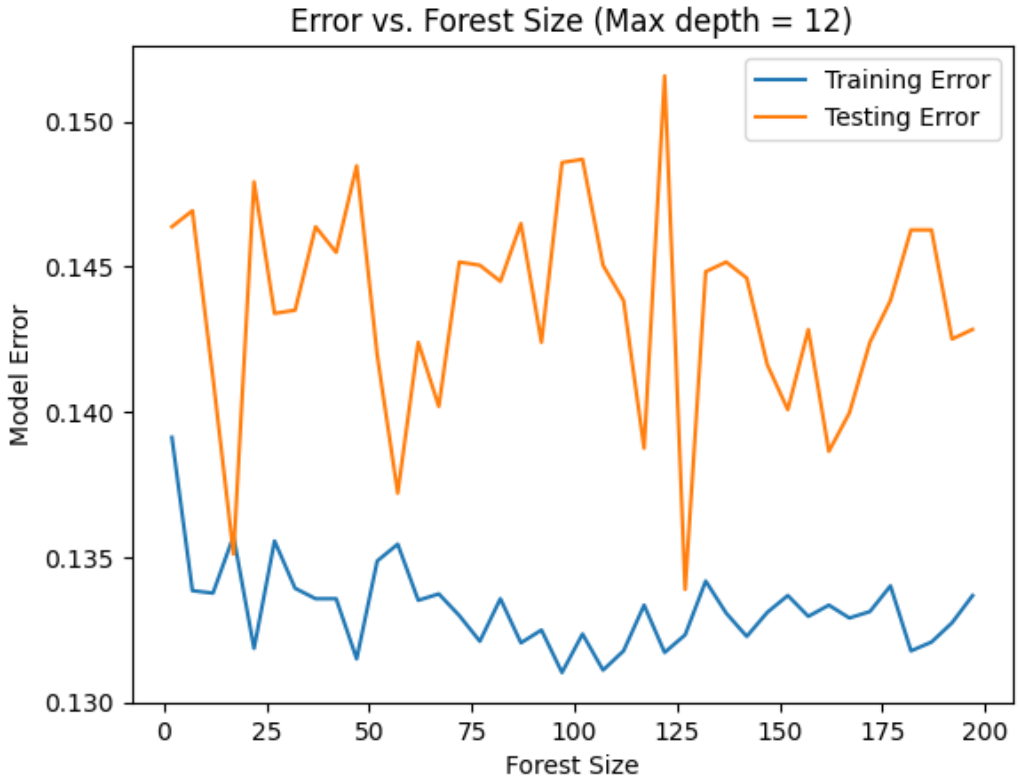

Figure 6: RF Error vs. Forest Size (max_depth = 12, dynamic set)

results shown in Figure **??** show the accuracy of models each of which was trained on the same dataset (static set) to minimize the randomness between iterations of **n_estimators**

It seems that changing the shuffled order of the dataset introduces a lot of variabiltiy in the accuracy of the resulting mode. This implies that some sequences of training examples are 'lucky', producing more accurate models.

The results of Figure **??** does not show an overfitting pattern to the same degree as was observed in the DTC experiments without pruning. The training and testing accuracy are different, but the difference does not appear to be increasing as the size of the forest increases.

A curve of best fit was fit to the accuracy data from this experiment, and it was found that the larger the size of the forest, the better the accuracy of the model. However, there is clearly a point of diminshing returns after about 40-50 trees. This is shown in Figure **??**. Future experiments will used 50 estimators, since this is the point of diminshing returns.

6.2 Maximum Number of Features, d'

Experiment 6: The data was split into 80% training and 20% testing sets. The maximum number of features a tree could consider for a split was varied from 1 to 104 (the maximum number of features in the data). For each value of **max_features** a new RFC model was trained on the training set and scored on the testing set. The results are shown in Figure **??**.

It seems that the benefit of including more features increases without bound. However, there is clearly a point of diminshing returns which begins after approximately 20 features. In this experiment the maximum accuracy was achieved with 90 features, so this value will be used in subsequent experiments. 90 features is larger than the default of $\sqrt{104} \approx 10$.

6.3 Bootstrap Sample Size

Experiment 7: This experient varied the size of the randomly selected bootstrap sample on which each tree is trained. The default size is the size of the training set. The data was split into 80% training and 20% testing sets. The **max_samples** parameter was varied from 0.01 to 1, representing the fraction of the training set which should be used to construct the bootstrap sample (where examples are drawn from the broader training set with replacement). The results are shown in Figure **??**.

The results of this experiment show that the accuracy of the model increases as the size of the bootstrap sample increases. However, there is a point of diminishing returns at approximately 3500 samples.

6.4 Minimum Number of Samples to Split

Experiment 8: This experiment varied the minimum number of samples required to split an internal node. The data was split into 80% training and 20% testing sets. The **min_samples_split** parameter was varied from 1 to 10,000 in steps of 10. Having a very small value for this parameter allows the tree to very finely tune to the training data, but may lead to overfitting to the particular training examples it has seen. Increasing the value of this parameter prevents the model from creating deeper trees which split on very small subsets of the data. The results are shown in Figure **??**.

The results show the overal random forest accurcy as a function of the minimum number of samples required for a tree to split on an internal node. The results show that there is no benefit in terms of model accuracy, to increasing the value of this paramter. The best accuracy is achieved at the minimum value of 2. For large models, there may be performance reasons for using a larger value for this parameter.

6.5 Out of Bag Error

Experiment 9: In this experiment a method for measuring the OOB error was implemented. The OOB error is the error of the model, using the majority vote among only those trees in the forest which were not trained on a particular example.

The OOB implementation used the provided sample code to retrieve a list of indices for each tree in the forest, specifying which examples were not used to train that tree. The predictions of each tree were collected and averaged to produce an overall prediction for the model.

OOB Implementation *as shown in a1_rf.ipynb*

```
def get_rf_prediction(rf, X):
    unsampled_i = find_unsampled_indices(...)
    estimators = rf.estimators_
    rf_predictions = []
    for i, x in enumerate(X):
        e_predictions = []
        for j, e in enumerate(estimators):
            if i in unsampled_i[j]:
                e_predictions.append(e.predict(x.reshape(1, -1)))
            else:
                pass
        avg_prediction = np.mean(e_predictions)
        if avg_prediction >= 0.5:
            rf_predictions.append(1)
        else:
            rf_predictions.append(0)
    return rf_predictions
```

```
def score_rf(predictions, y):
    correct = 0
    for p, y in zip(predictions, y):
        if p == y:
            correct += 1
    return correct / len(predictions)
```

The OOB error was measured and plotted against the size of the random forest. The results are shown below in Figure **??**. The OOB error does seem to broadly follow the same trend as the test error, particularly for higher numbers of estimators. However, the OOB error is always higher than the test error.

Neural Networks

7 NN Hyperparameters

Neural Networks (or Multi-Layer Perceptrons in scikit-learn vernacular) are a powerful class of machine learning models which can be used for classification. Neural Networks are highly generalizable and are able to learn complicated non-linar relationships, and can approximate any function arbitraryily well (with enough hidden-layer nodes).

The experiments in this section will primarily use the suggestion network architecture of 3 layers (1 input layer of 104 nodes, 1 hidden layer of a variable number of nodes, and one output layer, representing the predicted class label).

Scikit-learn provides a **MLPClassifier** class which is used in this section. The hyperparameters of the MLP Classifier which are of interest in the following experiments are:

- **n_estimators**: The number of trees in the forest.
- **max_depth**: The maximum depth of each tree.
- **max_features**: The max. number of features to consider for a split.
- **max_samples**: The max. number of samples drawn from the training set to train a single estimator.
- **min_samples_split**: The min. number of samples required to split an internal node.

8 Random Forest Classifier Experiments

8.1 Forest Size

Experiment 5:

Test

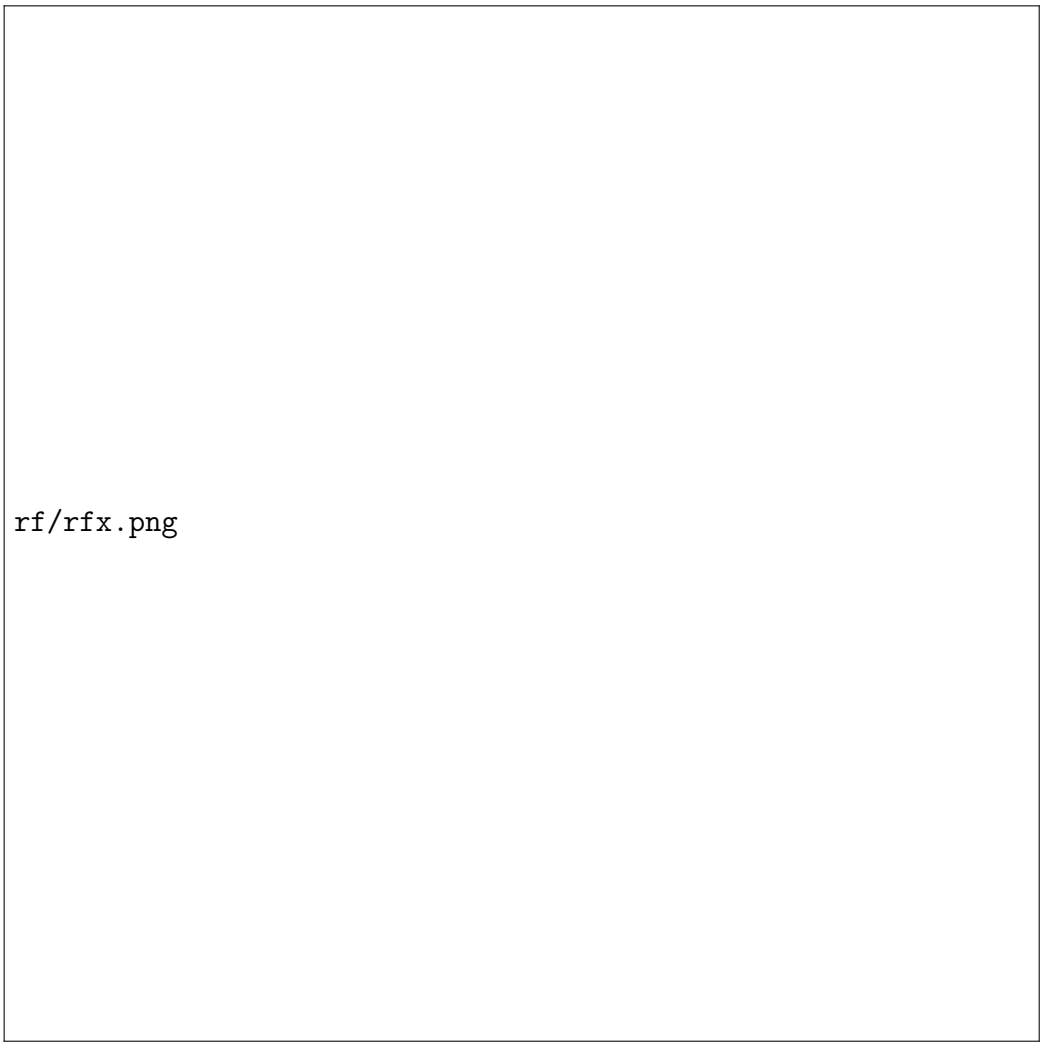

Figure 7: RF Error vs. Forest Size (max_depth = 12, static set)

```
maximum: 199.9984637007033 0.8583552232362794
```

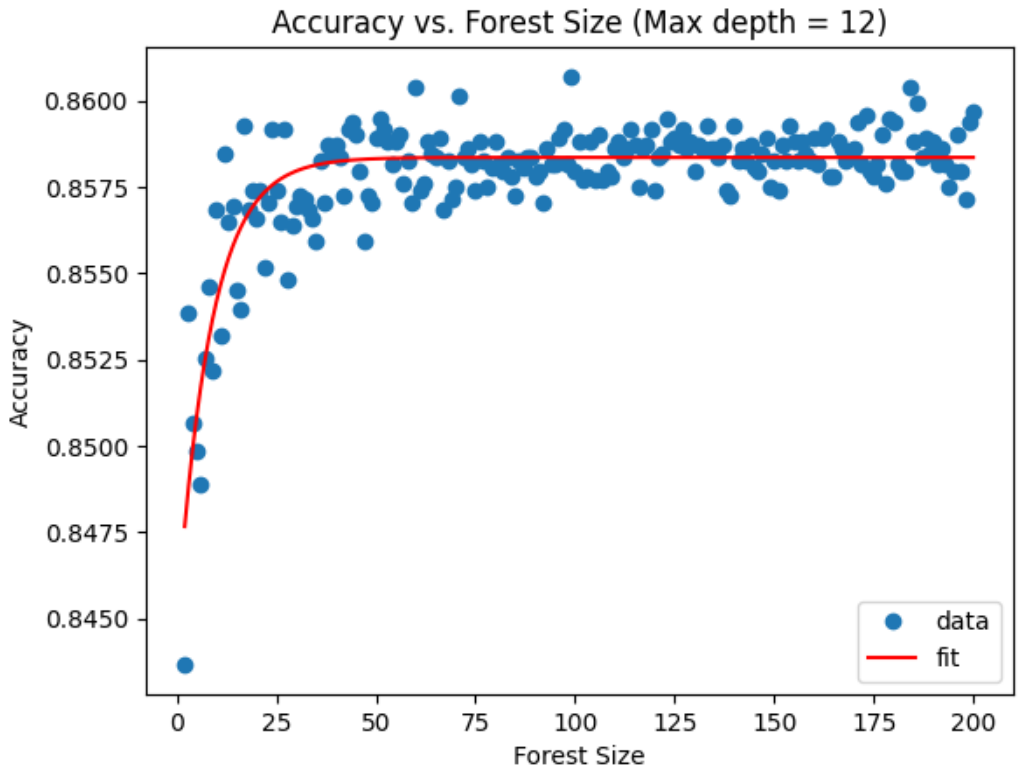

Figure 8: RFC Accracy vs. Forest Size (max_depth = 12, static set)

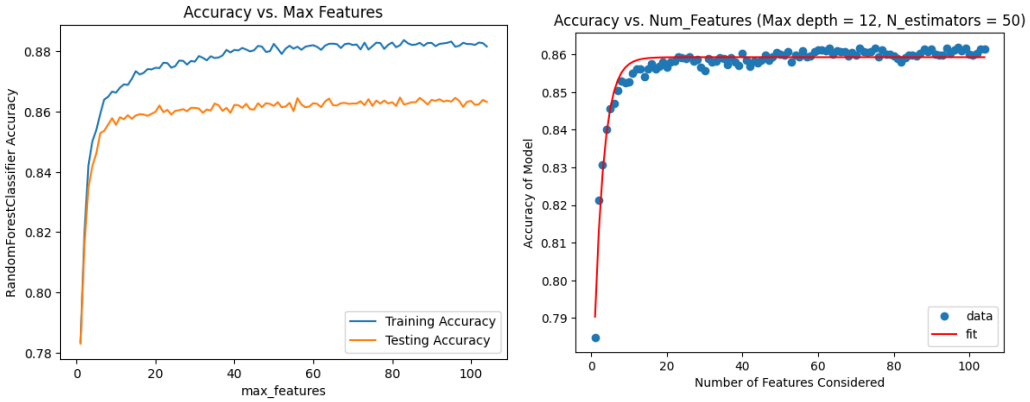

Figure 9: RFC Accuracy vs. Max. Features (max_depth = 12)

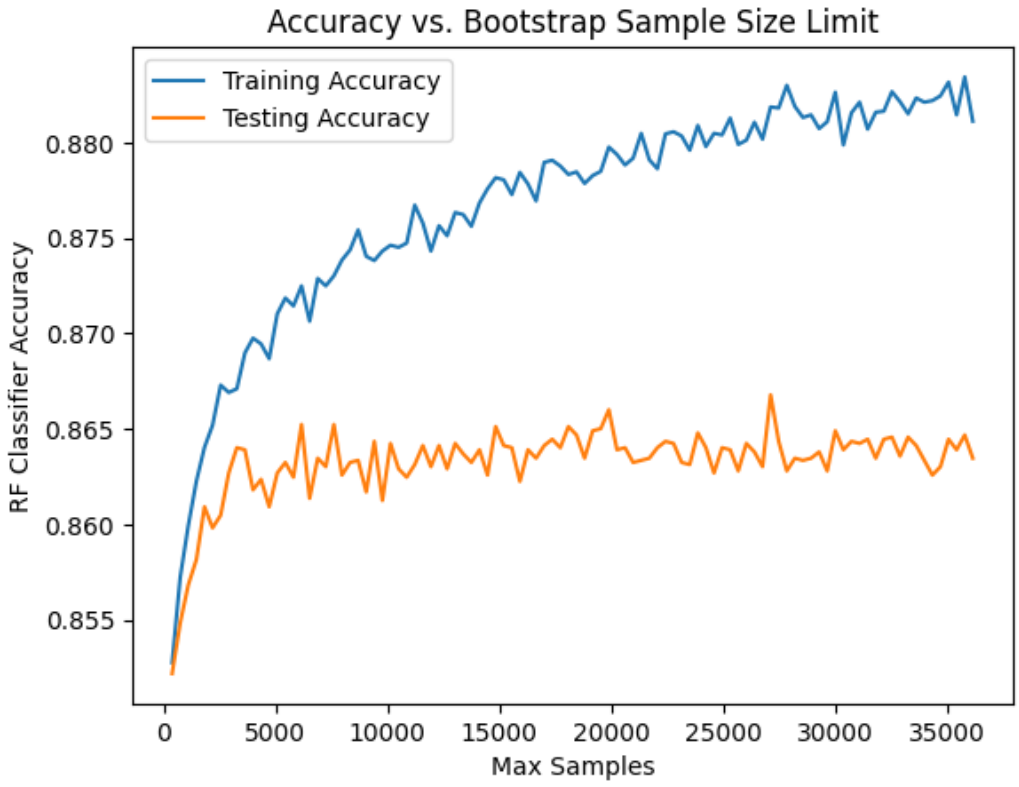

Figure 10: RFC Accuracy vs Bootstrap Sample Size

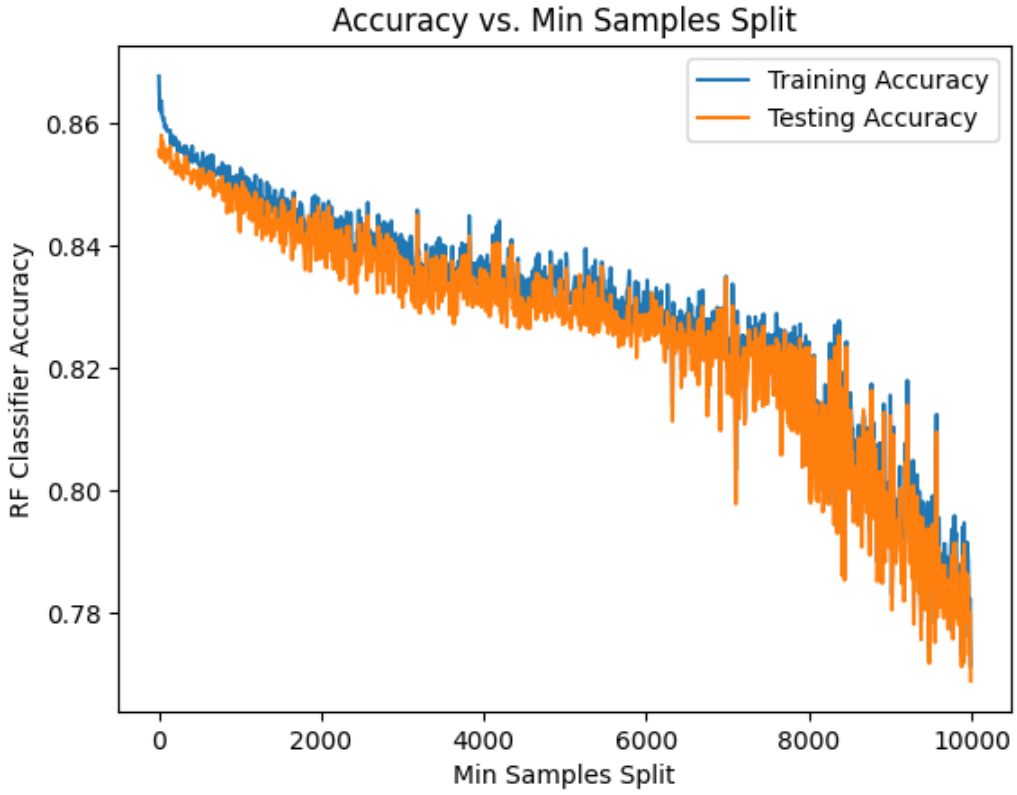

Figure 11: RFC Accuracy vs Min. # Samples to Split

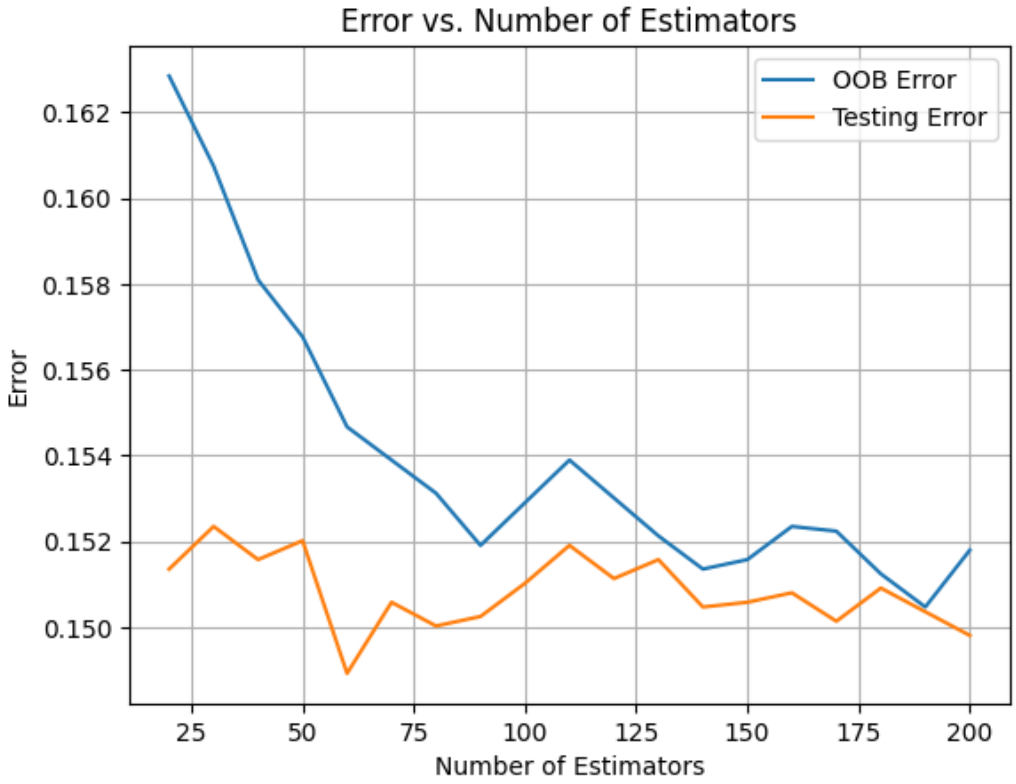

Figure 12: RFC OOB Error vs. Number of Estimators