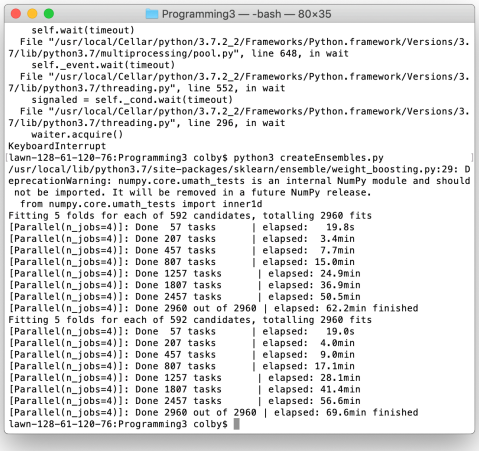
Colby Tobin
CS6220

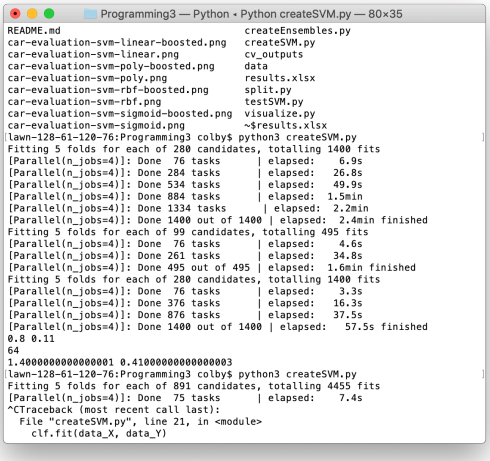
Programming Assignment 3: *Problem 2 – Supervised Learning*

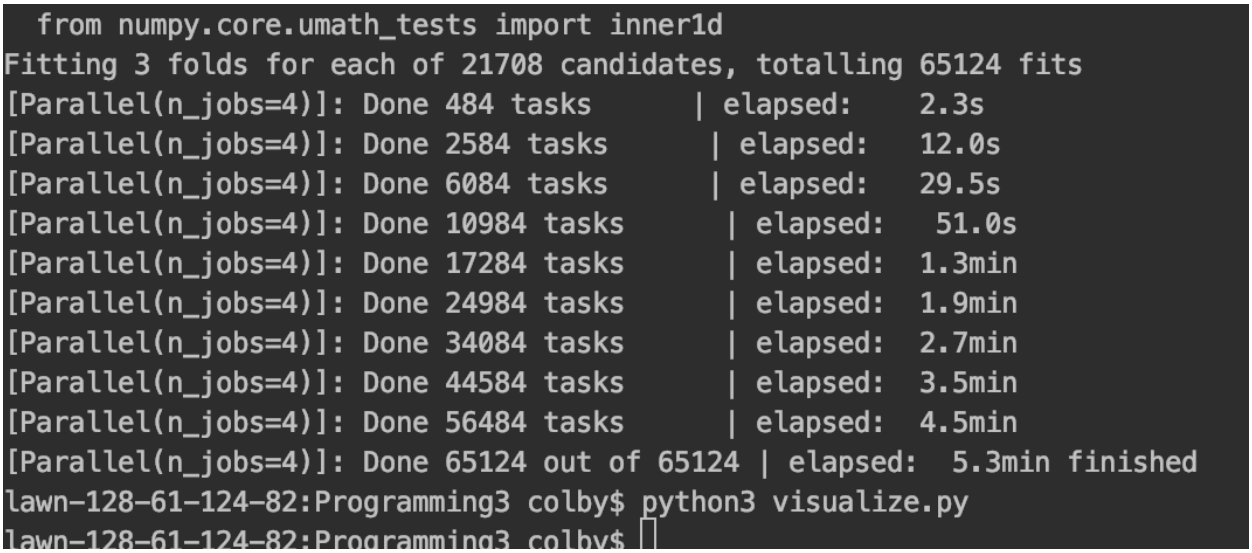
Link to Code: https://github.com/tobincolby/CS6220-Programming3

Link to Data: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

Environment Screenshots







Classification Problem

The first problem was classifying the acceptability of a car given the characteristics provided within the datasets. The acceptability of the cars was ranked as unacceptable,

acceptable, good, or very good. Each car also had a number of attributes associated with it, such as the purchase price, the maintenance level, the number of doors, the number of people it fits, the size of the luggage boot, and the safety estimation of the car. The purchase price and maintenance level of the car were categorized as low, medium, high, or very high. When processing the data, to make the training easier, I converted each of these values into a single value ranging from 1 to 4 respective of their categorization. The luggage boot size and safety estimation were categorized as low, medium, and high, so I converted each of these values into a single value ranging from 1 to 3 respective of their categorization. The number of doors and number of people categories used number representations except for their highest categorization. To account for this, I replaced the string to represent the highest classification with a number proportionally greater than the previous classification to make training easier. These attributes are all relevant because domain experts, at the time the data was collected, would describe the quality of a car based on these attributes. Regarding the data, there are 1728 instances of data with 6 attributes.

This dataset is complete, which means that all attributes are present for each of the instances. In addition, this data set results in a single output. In order to perform training and testing of the data, I split the data into a 70% training and 30% testing set. The 70/30 split was chosen based on previous findings using the dataset I selected. This dataset present itself as a difficult classification problem. The car evaluation dataset is solvable by human experts who understand the domain area. The instances represent all possible combinations of the attributes for the classification problem. For the dataset, all of the attributes of each instance of data can be represented as a discrete finite set of classes, meaning that the possible combination of data points is finite.

Ensemble Algorithm

For this programming assignment, the ensemble algorithm I decided to use was the Adaptive Boosting algorithm from *scikit-learn*. This is a boosting algorithm that fits the initial base classifier on the original dataset, and then makes additional copies of the classifier and fits the data on those classifiers, but it adjusts the weights of the incorrectly classified instances to focus more on difficult train cases. This algorithm was chosen for many reasons. The boosting algorithm helps reduce bias and variance in supervised learning problems. Additionally, this boosting algorithm helps convert weak learners into something similar to a “strong” learner. By manipulating the weights of the data, it allows for the algorithm to consistently correctly classify the easy instances, but it also places a larger emphasis on learning the harder instances, which ultimately makes the algorithm better in most cases than its respective base classifier. Additionally, the design of the boosting algorithm reduces the likelihood of overfitting, which is an important aspect in a supervised learning algorithm.

Measure of Success for Classification Problem

In this classification problem, there were 2 major things that were looked at to consider supervised learning algorithm’s success. The first is the classification accuracy. Each instance of data can be classified as one of 4 labels, and the classification accuracy on the test dataset was compared for each algorithm to determine which algorithm produced the best results in that

regard. To have an algorithm that could be considered successful would be one that has a better performance than random chance, so these algorithms need to perform better than 25% classification accuracy to be considered good learning algorithms. In addition to classification accuracy, the train time of each algorithm will considered as a measure of performance. With each of the supervised learning algorithms, there was a wide variety of train times that will be analyzed later in the paper.

Chosen Supervised Learning Algorithms

For this programming assignment, I have decided to use Support Vector Machines as the supervised learning algorithm. To vary the results of the experiment, I used various kernels for Support Vector Machine algorithm. These kernels included the RBF kernel, linear kernel, polynomial kernel, and sigmoid kernel. In addition to the kernel, SVM's have other hyper-parameters that required tuning. To do this, I used *scikit-learn's* GridSearchCV model selection algorithm with a 5-fold cross-validation to find the best hyper-parameters for each of these kernels. For the polynomial kernel, the C-value and degree-value were varied as hyper-parameters. For the RBF and sigmoid kernel, the C-value and gamma-value were varied as hyper-parameters. For the linear kernel, the C-value was varied as the hyper-parameter for that SVM. Once the best SVM for each kernel was determined, they were then used as the base classifier for the Adaptive Boosting algorithms, which would allow comparison with ensemble algorithms.

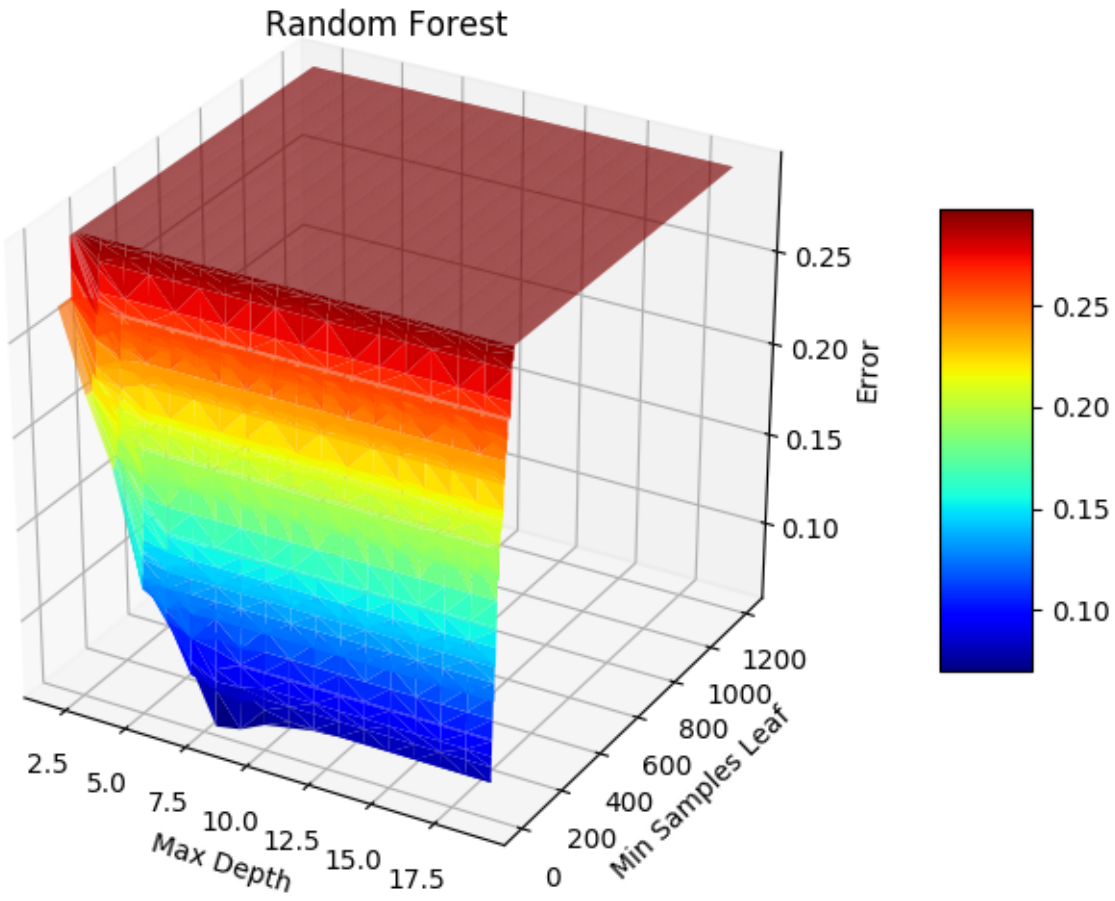
In addition to SVM's, I also performed an experiment with using *scikit-learn's* RandomForestClassifer to see how that compared with the Adaboost classification algorithm. For the RandomForestClassifier, I chose the number of estimators to be 5, per the assignment description. In addition to number of estimators, there are other hyper-parameters that exist for the RandomForestClassifier that need to be tuned as well. For this experiment, the maximum depth and minimum # of samples required to be a leaf node were the 2 hyper-parameters that were varied. Like the SVM algorithms, I used *scikit-learn's* GridSearchCV model selection algorithm with a 5-fold cross-validation to find the best hyper-parameters for the RandomForestClassifier.

Data/Results

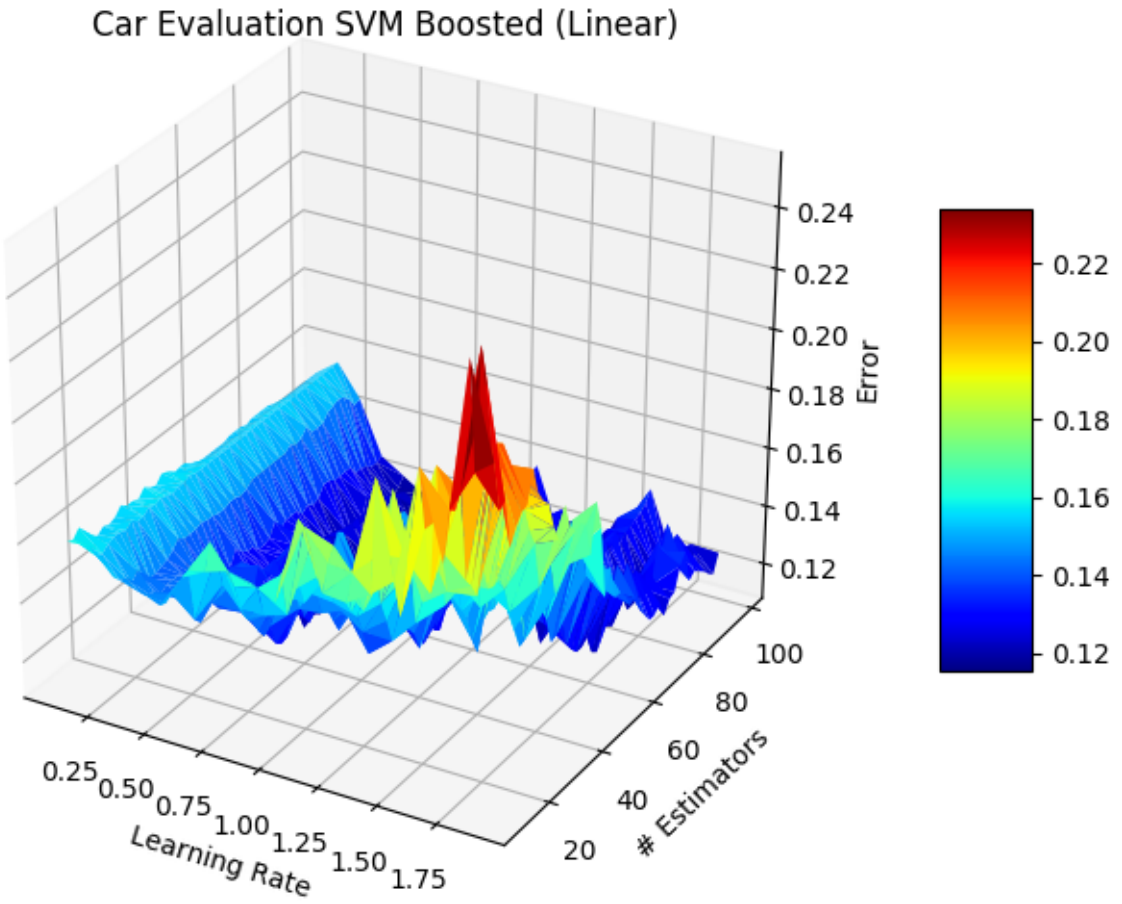
The data and performance results from this experiment are saved in the excel document attached to this assignment.

Observations and Conclusions

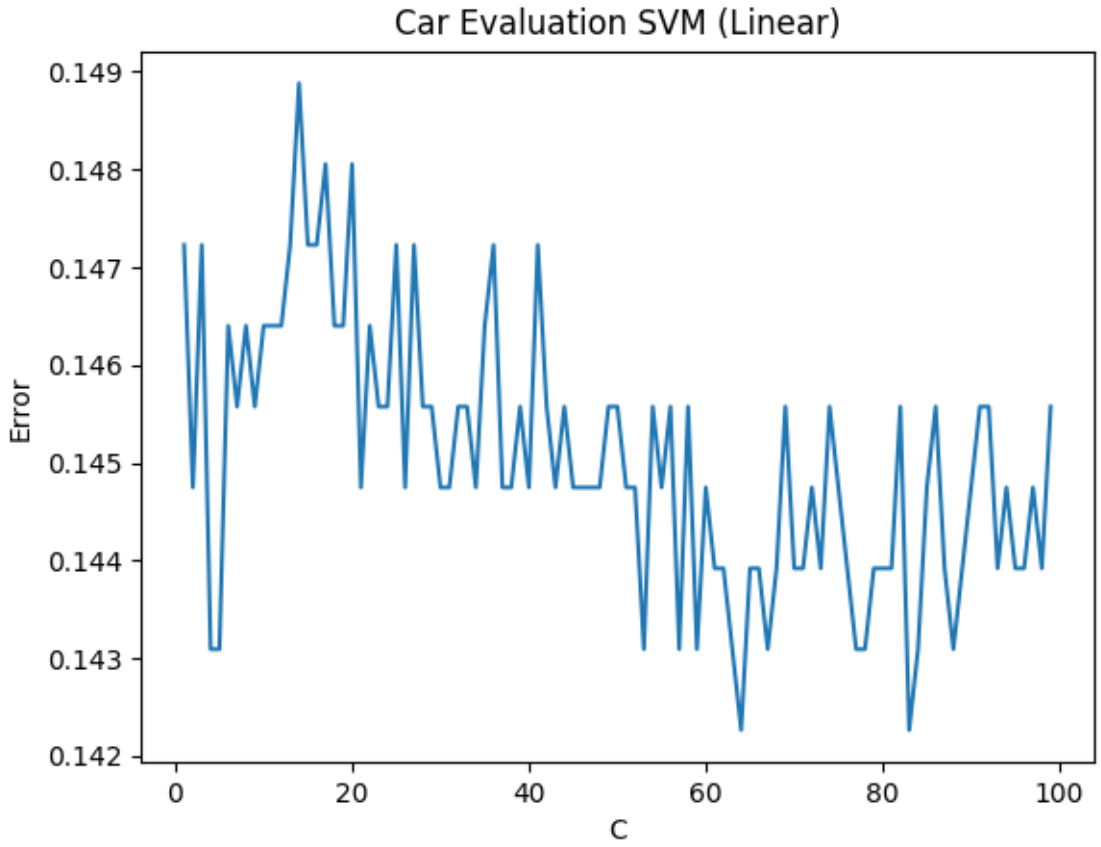
This experiment provided many insights to the workings of the different classifiers that I used. For the observations and conclusions section, I will first analyze the hyper-parameter tuning for each of the classifiers, then I will analyze their runtime performance on training and test datasets, and then I will provide my overall conclusions from the experiment.



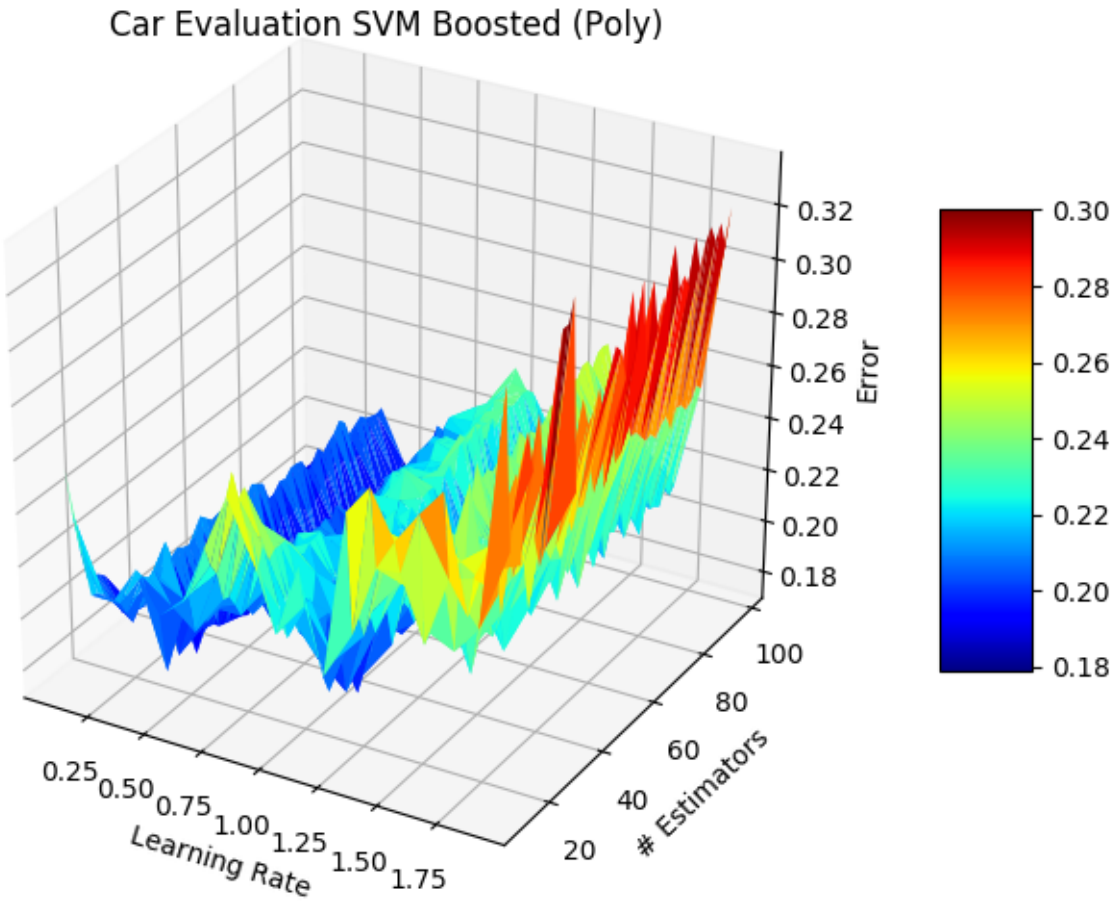
The above graph demonstrates a visual of the hyper-parameter turning for the random forest classifier. For the random forest classifier, the max-depth of the random decision trees and the minimum # of samples required to be a leaf node were varied for these parameters. As seen by the graph, the ideal maximum depth for this dataset is 8. As the maximum depth becomes larger or smaller than 8, the model begins to overfit the data resulting in a worse performance. With regards to minimum samples leaf node, it can be seen that the ideal minimum samples is 1 to be a leaf node. As the # increases to 1200, it can be seen that model begins to overfit the data and have a worse performance, so 1 is the ideal number.



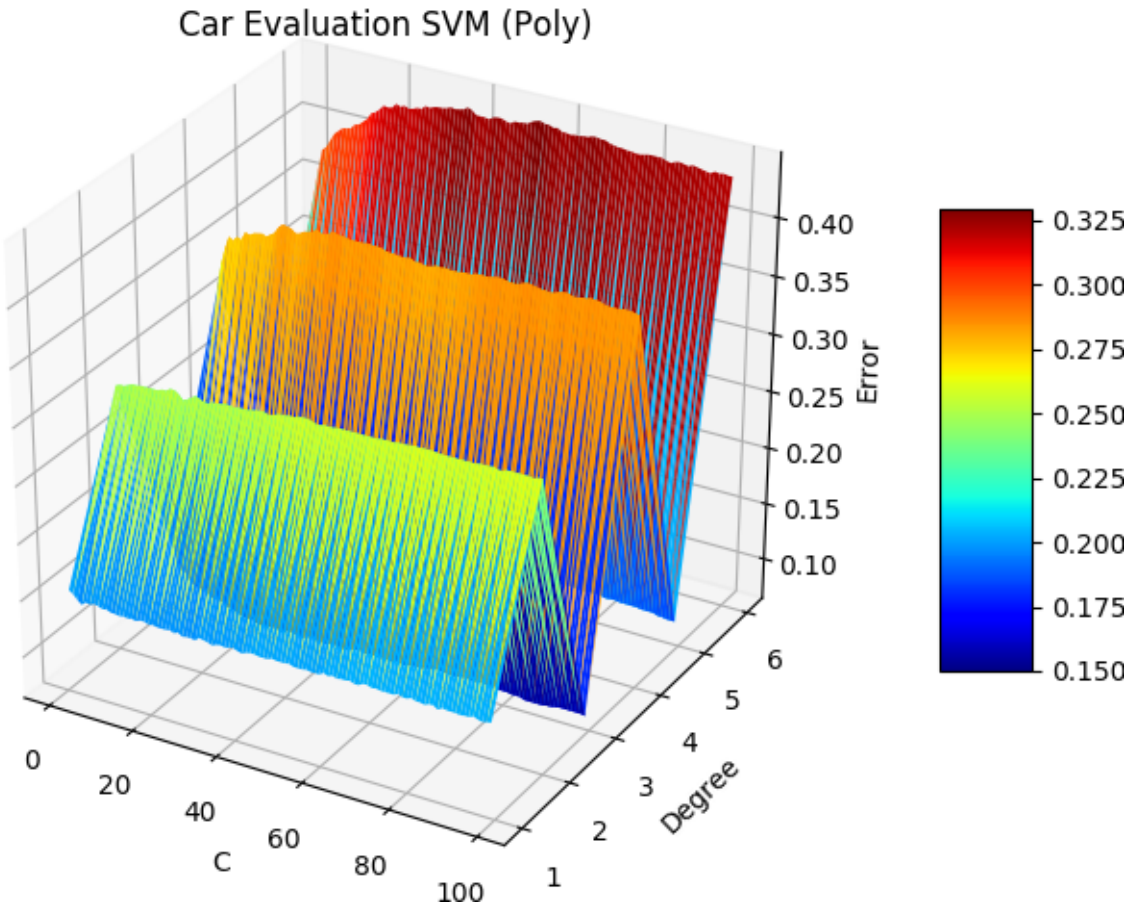
The above graph depicts the hyper-parameter tuning for the boosted SVM with a linear kernel. In this classifier, the learning rate for the AdaBoost algorithm and the # of estimators were the varied parameters. The ideal parameters for this SVM has the # of estimators around 68 and the learning rate around 1.7. As the learning rate decreases to 0, it can be seen that the performance worsens as it overfits the data by not learning as much from previous iterations of the algorithm. As the number of estimators increases from 68, it can be seen that there is a significant increase in error rates as the increased number of estimators results in a model that is designed to overfit the data. As the number of estimators decreases, it can be seen that the model performs more similarly to a non-boosted version of the kernel, which is why 68 is the ideal number.



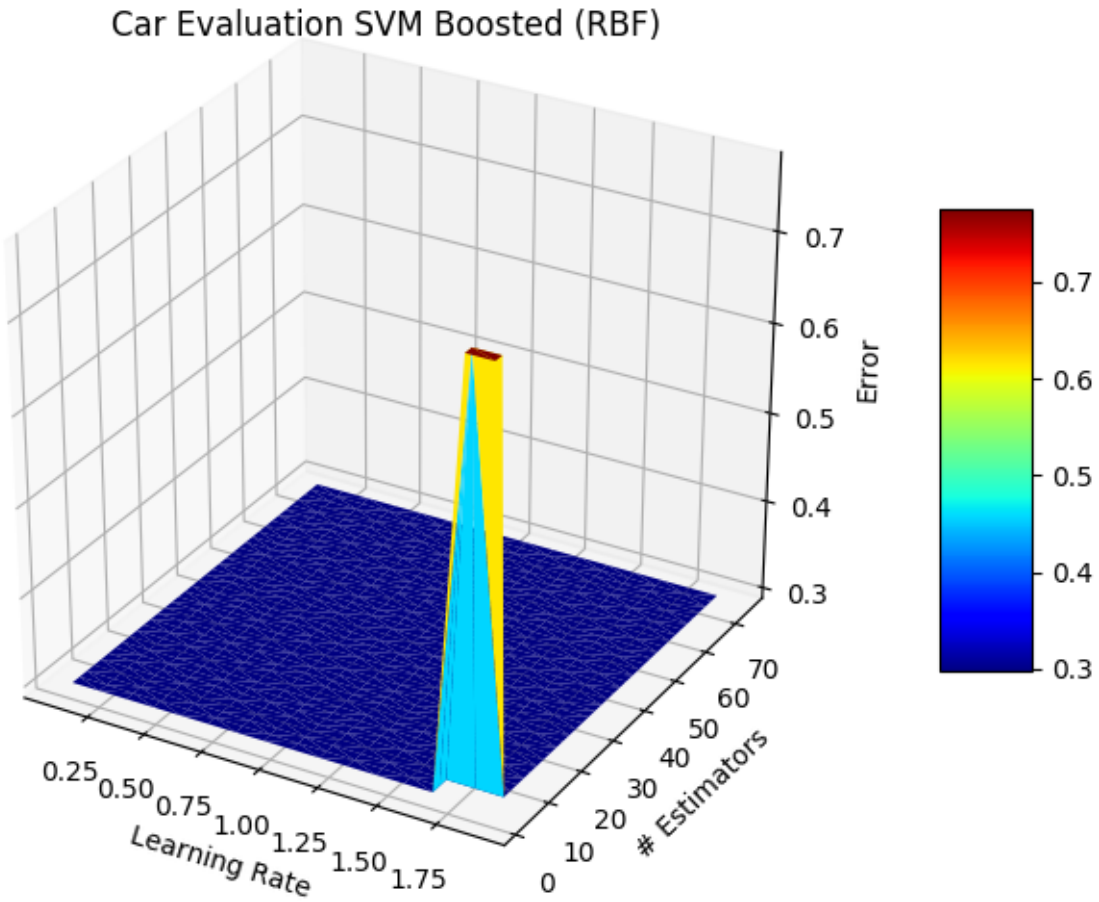
The above graph depicts the hyper-parameter turning for the SVM with a linear kernel. The only value that was varied for this was the C-penalty because there were no other meaningful parameters to change. Based on this graph, it can be seen that the ideal C-penalty is around 64. Values lower and higher than 64 result in a worse performance for the SVM classifier, so it makes sense that the ideal SVM linear classifier has a C-penalty of 64.



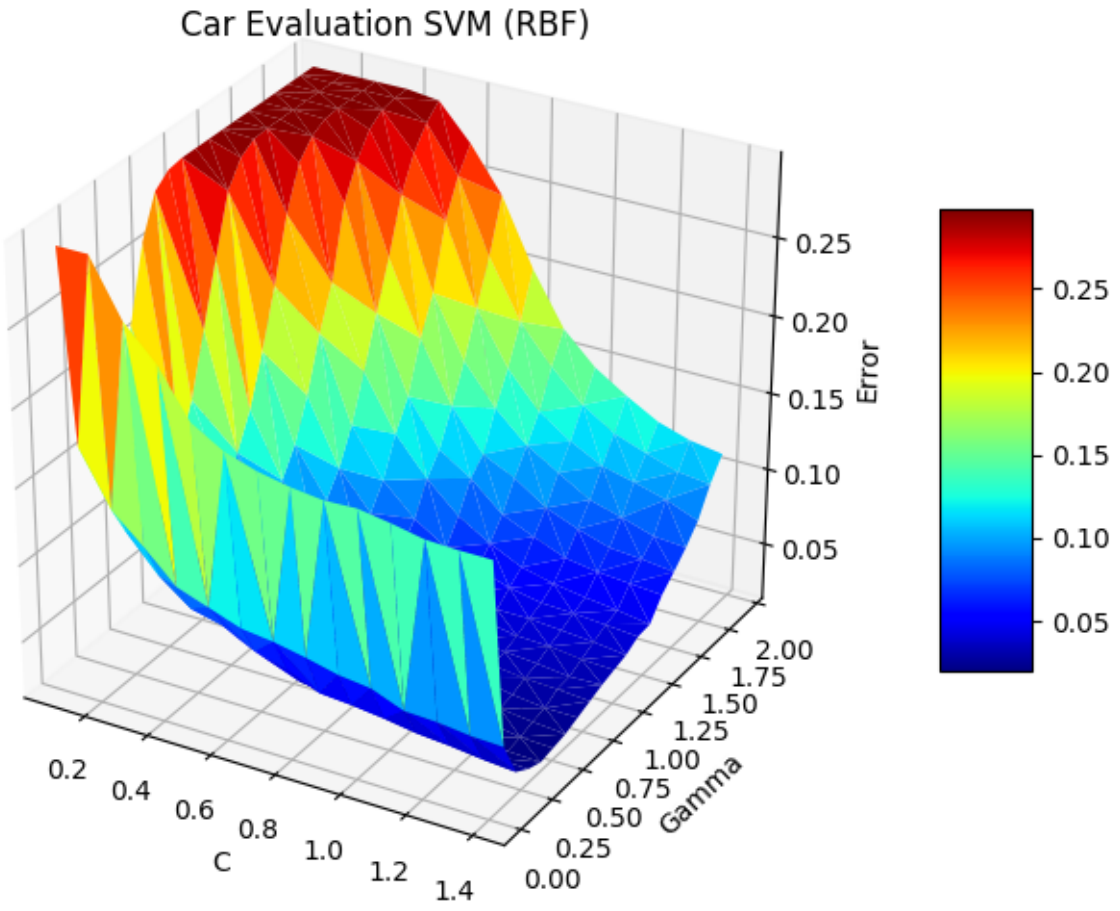
The above graph depicts the hyper-parameter tuning for the boosted SVM with a polynomial kernel. In this classifier, the learning rate for the AdaBoost algorithm and the # of estimators were the varied parameters. Based on the above graph, it can be seen that the ideal # of estimators is around 68 and the ideal learning rate is around 0.6. As the number of estimators increases, the model begins to overfit the data, whereas when the number of estimators decreases, it seems to behave more similarly to the non-boosted version of the SVM, so 68 is the ideal number of estimators for this boosted kernel. With regards to the learning rate, as the learning rate increases, it results in a model that overfits the training data more, which is evident in the sharp increase in error rates as seen by the graph.



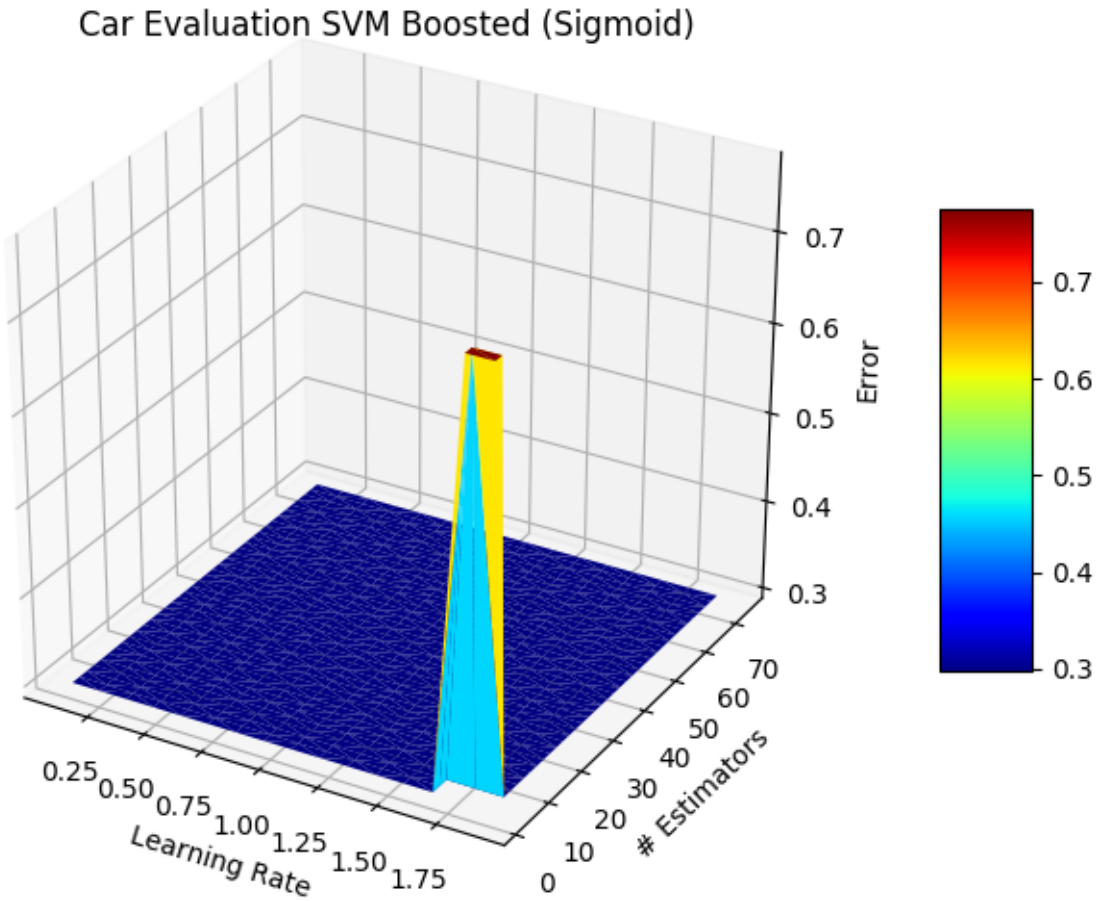
The above graph depicts the hyper-parameter tuning for the SVM with a polynomial kernel. For this SVM, the varied parameters were the C-penalty and degree of the polynomial. Based on the above graph, it can be seen that the ideal degree for the SVM is 3. As the degree is increased or decreased, it can be seen that the model performs worse. Additionally, it seems as though the polynomial kernel with odd degrees seem to perform better than the even degrees, but a degree of 3 is optimal. Additionally, based on the graph, it can be seen that C-penalty doesn't have a big effect on the overall performance of this model. However, from the graph, we can determine that ideal C-penalty is around 27.



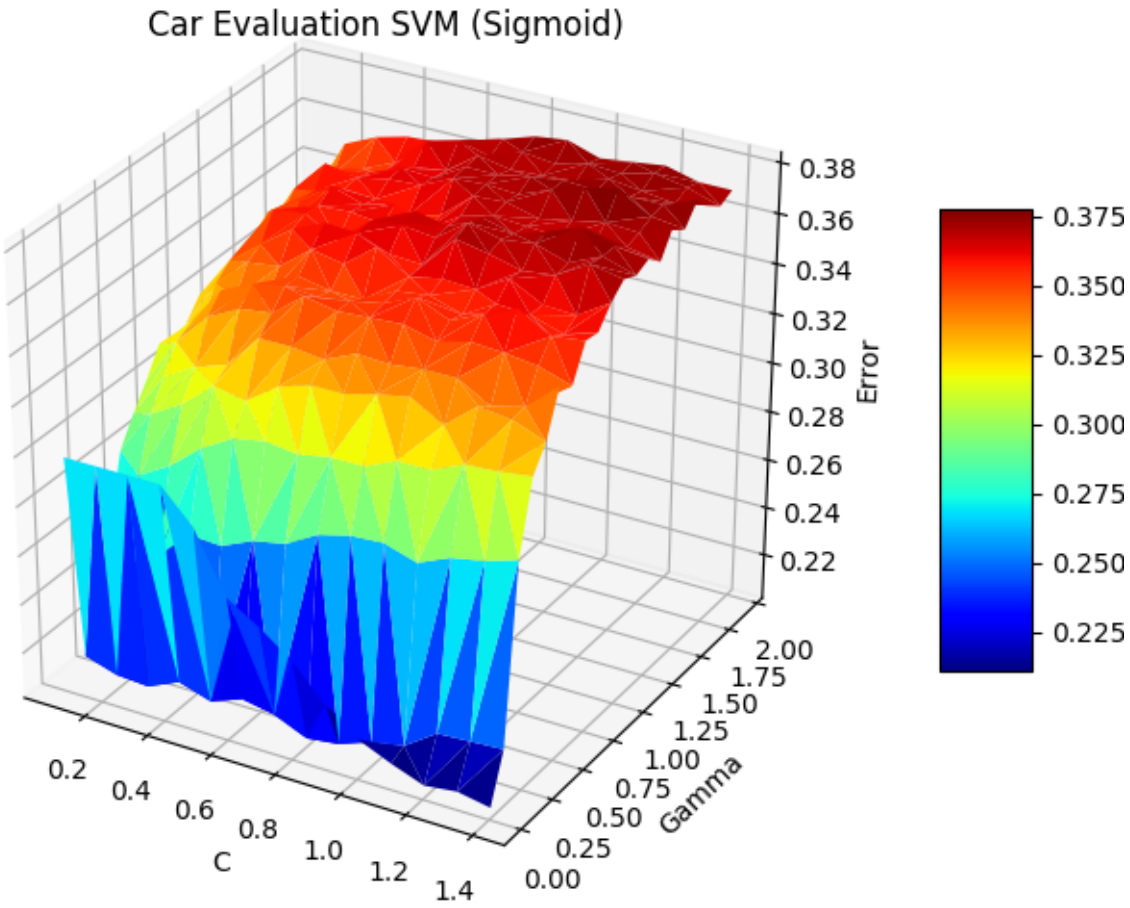
The above graph depicts the hyper-parameter tuning for the boosted SVM with a RBF kernel. For this classifier, the tuned parameters were the learning rate and the number of estimators for the AdaBoost algorithm. Based on the graph, it shows some interesting results as the overall performance of the algorithm does not really depend on the different parameters for the algorithm. It all seems to perform similarly with worse results than the non-boosted version of this SVM. With this in mind, there might have been some sort of error that occurred when generating the model or this could conclude that the SVM with an RBF kernel does not work well when boosted using the AdaBoost algorithm. With more time for these experiments, I would have liked to determine the cause for these results in the above graph, but due to computational constraints and time constraints, that is not something that could have been done.



The above graph depicts the hyper-parameter tuning for the SVM with the RBF kernel. For this kernel, the C-penalty and gamma-value were varied as the hyper-parameters for the SVM. Based on the above graph, it can be seen that the ideal C-penalty is around 1.4, and the ideal gamma-value is around 0.41. As the gamma-value increases or decreases from 0.41, it can be seen that the model performs much worse on the data, and might overfit the data more than when it is 0.41. With regards to the C-penalty, it can be seen that as the C-value decreases from 1.4, the model performs much worse on the dataset, so this shows that a larger C-penalty provides better results in the case of the RBF kernel.



The above graph depicts the hyper-parameter tuning for the boosted SVM with a Sigmoid kernel. For this classifier, the tuned parameters were the learning rate and the number of estimators for the AdaBoost algorithm. Based on the graph, it shows some interesting results as the overall performance of the algorithm does not really depend on the different parameters for the algorithm, which is very similar to the boosted RBF SVM. It all seems to perform similarly with worse results than the non-boosted version of this SVM, and performs with almost identical results to the boosted RBF classifier. With this in mind, there might have been some sort of error that occurred when generating the model or this could conclude that the SVM with a Sigmoid kernel does not work well when boosted using the AdaBoost algorithm. With more time for these experiments, I would have liked to determine the cause for these results in the above graph, but due to computational constraints and time constraints, that is not something that could have been done.



The above graph depicts the hyper-parameter tuning for the SVM with a sigmoid kernel. For this kernel, the C-penalty variable and the gamma-value variable were varied for this experiment. Based on the above graph it can be seen that a C-penalty value of around 0.8 is ideal, and a gamma-value around 0.11 results in the best performance. As the gamma-value increases from 0.11, it can be seen that the model begins to strongly overfit the data as the error significantly increases with an increased gamma-value. With regards to the C-penalty, it doesn't seem like it has as much effect on the performance of the sigmoid kernel classifier because the error is fairly consistent across the values of the C-penalty for each gamma-value; however, you can see some slight increases in the error of the model as the C-penalty value increases and decreases from 0.8.

Performance Comparisons

For this analysis, I will be analyzing and comparing the data from the excel sheet that is attached to this submission.

First, I would like to compare the linear kernel SVM with its boosted counterpart. The SVM with the linear kernel performed with an accuracy of 84%. It took the SVM 96 seconds to train and obtain the best classifier. On the other hand, the boosted linear SVM performed with an

accuracy of 87%, but it took the boosted SVM 3,492 seconds to train and obtain the best classifier. This train time for the boosted version of this algorithm is significantly longer than the original classifier. This makes sense because in the boosted algorithm, it must actually train multiple linear SVM's as a part of the overall boosted SVM, which gives rise to why it took longer to train. However, the boosted SVM for the linear kernel did perform better than the regular linear kernel by around 3%. This was expected as the AdaBoost algorithm places more emphasis on the harder instances, so it can improve its classification accuracy. However, with the sheer extra space and time required to train the boosting algorithm for this kernel, it might not have been worth it because it only resulted in a very slight increase in accuracy.

Next, I would like to compare the poly kernel SVM with its boosted counterpart. The SVM with the polynomial kernel performed with an accuracy of 89.7% and it took the SVM 756 seconds to train and get the best classifier. With the boosted polynomial SVM, it performed with an accuracy of 78.9% accuracy, and it took the 4254 seconds to train this classifier. This result is fairly unexpected, as it is typically seen that boosting improves classification. However, in some cases, overfitting can occur when using some boosting algorithms due to the usage of multiple classifiers combined into one model. It is not common, but it does occur, which is likely what we are seeing here with these results. Overall, based on this information, clearly, it does not make sense to use the ensemble algorithm to boost the polynomial SVM as it results in a worse classification accuracy and takes a much longer time to train as well.

Next, I will compare the RBF kernel SVM with its boosted counterpart. The SVM with the RBF kernel performed with an accuracy of 97.7% accuracy, and it took the SVM 144 seconds to train. On the other hand, the boosted RBF SVM performed with an accuracy of 69.6% accuracy and took 3732 seconds to train. Based on the cross-validation results described earlier in this analysis, I believe there might have been some errors regarding the training of the boosted RBF SVM, which could potentially be a reason for these results. The lack of time and computational constraints made it difficult to determine the cause for these results. However, if we treat these results as valid, it can be seen that the boosted RBF SVM clearly resulted in an overfitting of the data, as its classification accuracy decreased significantly, which points to the idea that boosting would not make sense for this SVM kernel.

Now, I will compare the sigmoid kernel SVM with its boosted counterpart. The SVM with the sigmoid kernel performed with an accuracy of 75.5% accuracy, and it took the SVM 57.5 seconds to train. With the boosted sigmoid SVM, it performed with an accuracy of 69.6% and took 4176 seconds to train. Similar to the boosted RBF SVM, I believe there might have been some errors regarding its training based on the cross-validation results discussed earlier. Like before, if we treat these results as valid, it can be seen that the boosted sigmoid SVM seems to overfit the data in comparison to its regular counterpart because it results in a slightly lower classification accuracy. This leads to the idea that the sigmoid SVM should not be used in conjunction with a boosting algorithm due to its longer train times and its worse performance overall.

Finally, I would like to analyze the random forest classifier. The random forest classifier performed with an accuracy of 91.5%, and it took the random forest classifier 318 seconds to train. When comparing this to the SVM's analyzed above, it can be seen that the random forest

classifier does better than most of the SVM's with this classification problem. The only one it performs worse than is the RBF SVM, which has an accuracy that is 6.2% higher. When comparing it to the best classifier (RBF SVM), it also took almost twice as long to train, but this could stem from the fact that it had to iterate over 65,000 combinations of hyper-parameters to get the best random forest classifier. Additionally, with regards to space complexity, an RBF SVM takes up much less space than a random forest classifier because the random forest requires the model to keep track of 5 decision trees and their weights. Overall, because of this, it makes sense that an RBF SVM would be used over the random forest classifier for this classification problem due to its slightly better performance, less space requirements, and quicker train times.

Overall Conclusions

Based on the data gathered and the performance of the various SVM's, it can be seen that in certain instances, boosting improves overall classification accuracy and can improve a model's performance; however, that is not always the case as seen with the RBF, sigmoid, and polynomial kernels, where the boosting algorithms resulted in a worse performance overall. Additinally, when looking at this problem specifically, without boosting any of the SVM's, the classifiers were still performing "well" on the data. They were all performing well above random chance, which demonstrates that these models are still good. Even when boosting was applied and the classification accuracy decreased, they were still performing way above random chance, so the models were still doing well on the data.

Out of the many supervised learning models, SVM's are one of the more complex models due to the way they are trained and the way they classify instances of data in comparison to K-NN or Decision trees. Because of their complex nature, this could give rise as to why boosting doesn't necessarily improve their classification accuracy. They are already complex by themselves without introducing multiple weighted estimators to aid in classification. Sometimes, the simplest model is the correct model, and overcomplicating a model could result in overfitting and worse results, which is what the experiments point to.

Additionally, SVM's are very complex, which is why they have a very long training time in comparison to other supervised learning models. When taking this into consideration, boosting on a model that already takes a long period of time to train will only make training times exponentially longer as seen in the data. When comparing the tradeoffs of greatly increased training time with the benefits of a slightly better classification from the boosting, it seems as if boosting in this case is not necessary.

Lastly, each of these supervised models perform according to the problem and data itself. In some cases, boosting might result in better classification accuracy, and certain SVM kernels might perform better than others. In this specific problem, it can be seen that boosting did not help with the classification accuracy overall.