Colby Tobin (ctobin30)
Professor: Charles Isbell
February 10th, 2019

CS 4641 Supervised Learning Analysis

Introduction

This paper compares different supervised learning algorithms on 2 different classification problems. The data for these problems were found on the UC Irvine database of machine learning problems.

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.

The next classification problem deals with classifying tic tac toe games as a win or a loss for "x". The wins are represented as a positive one and a losses are represented as a negative one. The 9 attributes of this data set represent the different spaces within a tic-tac-toe endgame board, where each attribute can take on the role of "x", "o", or a blank space. To account for this, I manipulated the data, so that an "o" is a 0, a "x" is a 1, and a blank space is 2 for each of the 9 attributes. The data has been classified as a win or lose based on the rules of tic-tac-toe. In this data, there are 958 instances with 9 attributes for each instance of data.

Each of these data sets are complete, which means that all attributes are present for each of the instances. In addition, these data sets also result 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 datasets I selected. These datasets present themselves as difficult classification problems. The car evaluation dataset is solvable by human experts who understand the domain area, whereas the tic-tac-toe classification problem can be solvable by humans who understand the rules to tic-tac-toe. When comparing the datasets, it can be seen that the car evaluation data set has almost twice as many instances of data, but the tic-tac-toe dataset has more attributes per instance of data. In each dataset, the instances represent all possible combinations of the attributes for each respective classification problem. For each of these datasets, 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. These different datasets result in varying performances and error rates in the various supervised classification algorithms.

Decision Trees

For the decision tree algorithm, I used the *scikit-learn DecisionTreeClassifier* algorithm. The algorithm that this classifier utilizes is CART, which is where the Decision Tree is constructed as a binary tree. Because both of the datasets are composed of a finite discrete set of values, the data did not need to be normalized. The method used to split the data was the Gini Impurity. In order to tune the hyperparameters, I used *GridSearchCV* from *scikit-learn* to perform a 5-fold cross validation against the varied hyperparameters that will be discussed below. Below is the graph that shows the error rates of the decision tree model from *GridSearchCV*.

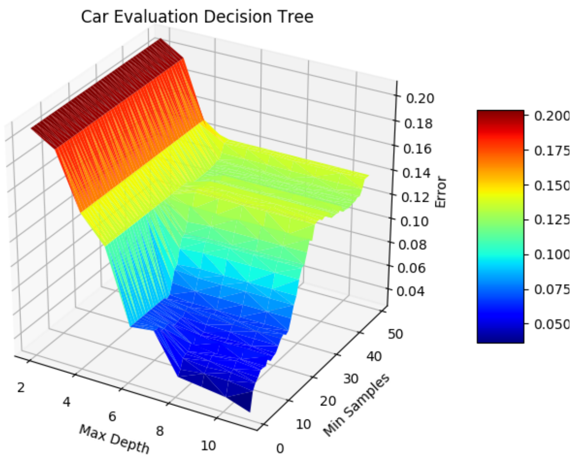

Figure 1: Car Evaluation Decision Tree

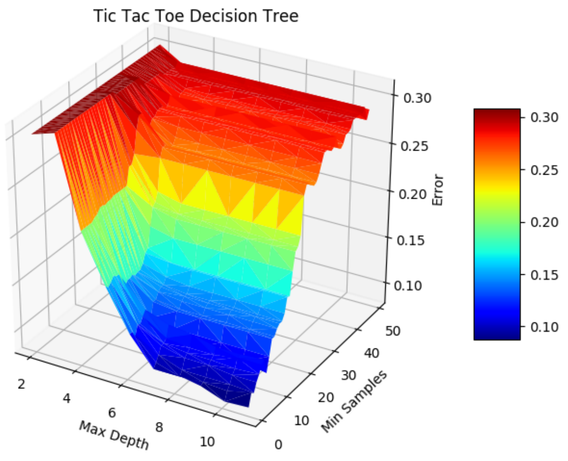

Figure 2: Tic Tac Toe Decision Tree

Originally, by setting the algorithm to all default parameters, it allows the algorithm to run without pruning. This provided me with a set of worst case parameters to use for the range of values for the changing hyperparameters. The first hyperparameter that I varied was the minimum number of samples required to be at a leaf node. With this parameter, it can be seen that error decreases proportionally in both datasets as the min-samples begins to decrease. The models seem to overfit the data as the minimum number of samples required to be a leaf node increased to the maximum value in the range. The other hyperparameter varied in this algorithm was the maximum depth of the decision tree. The maximum value of the range for this hyperparameter was selected from the unpruned decision tree. By adding a maximum depth to the decision tree, it acts as a form of pruning by forcing the tree to gather the most amount of data at the top of the tree, which acts as a way to reduce overfitting of the data. Within both datasets, increasing the max depth of the decision tree demonstrates a drastic decrease in error after the depth of 3 nodes, but the models begin to slightly overfit the data as the maximum depth is approached, since pruning no longer occurs in this situation.

Naturally, the decision tree algorithm was trained very quickly because of its simple structure as a binary tree. When comparing the 2 graphs of the 2 different datasets, they present very similar hyperplanes for the shape of the graphs, which presents the idea that decision trees perform similarly amongst these datasets even though the car evaluation dataset shows slightly less training error from the cross validation.

Neural Nets

The neural net algorithm that I implemented was the *MLPClassifier* from *scikit-learn*, which constructs a neural net based on multi-layered perceptrons. The solver for this algorithm was the *lbgfs*, which is recommended by *scikit-learn* as it maximizes the log-loss function of the model. The activation function for this model is represented as a rectified linear unit function, which returns the maximum between 0 and its input. Outside of the varied hyperparameters, the rest of the *MLPClassifier* parameters were kept at their default value. Below, I have presented the graphs that demonstrate the changes in error rate based on the varying hyperparameters from *GridSearchCV*.

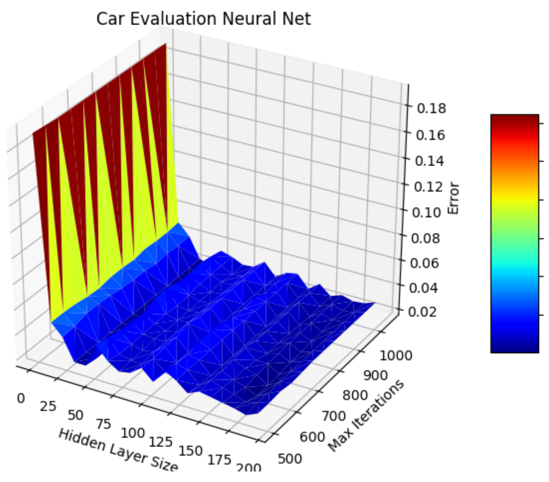

Figure 3: Car Evaluation Neural Net

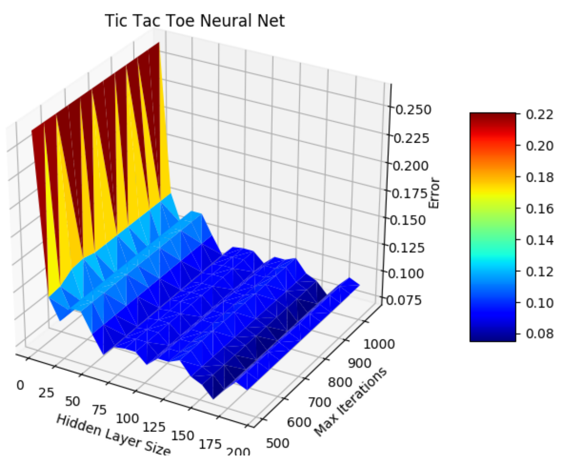

Figure 4: Tic Tac Toe Neural Net

In this algorithm, the first varied hyperparameter was the number of hidden layers. The number of hidden layers ranged from 0 to 200. 200 was chosen as the upper bound for this because of the processing power constraints on my computer. When looking at the car evaluation dataset, it can be seen that ideal hidden layer size was around 35 hidden layers. After this number of hidden layers, the error hovered around a constant rate above the optimal number of hidden layers. When looking at the tic-tac-toe dataset, the ideal number of hidden layers was around 165 hidden layers, which presents the lowest error rate. After this number of hidden layers, the error rate begins increasing again, which can be attributed to overfitting due to the increased number of hidden layers and increased complexity. The other hyperparameter that was varied was the maximum number of iterations allowed for the neural net to converge. Because this is not a stochastic solver, the number of iterations means the number of gradient steps required by the solver to converge. By increasing this number, it gives the solver more time to converge to a given set of values. In theory, increasing the number of maximum number of iterations would result in lower error rates because the solver has more time to converge; however, for both datasets based on the graphs, the maximum number of iterations does not impact the error rate significantly, as the error rate stays relatively constant across maximum number of iterations for each level of hidden layer size.

The neural net model produced error rates that were less than 1% lower than the error rates presented in the decision tree models. With this in mind, this presents the idea that neural nets might be too complex for these datasets. The model took a very long time to train and resulted in only a slightly better performance compared to decision trees despite its high level of complexity.

Boosted Decision Trees

For the boosted decision trees, the *AdaBoostClassifier* from *scikit-learn* was implemented. The Adaptive Boost Classifier utilizes a base estimator, in our case a decision tree, that originally fits the dataset onto the base estimator, but later begins increasing the weights of the incorrectly classified instances, so the classifier begins focusing on those incorrect classifications. Like mentioned above, the base estimator used for the Adaptive Boost algorithm was a decision tree that had a maximum depth and minimum number of samples to be a leaf node from the best decision tree determined in the previous decision tree section. Below, I have presented the error rates of the AdaBoost algorithm based on the hyperparameters that were varied using *GridSearchCV*.

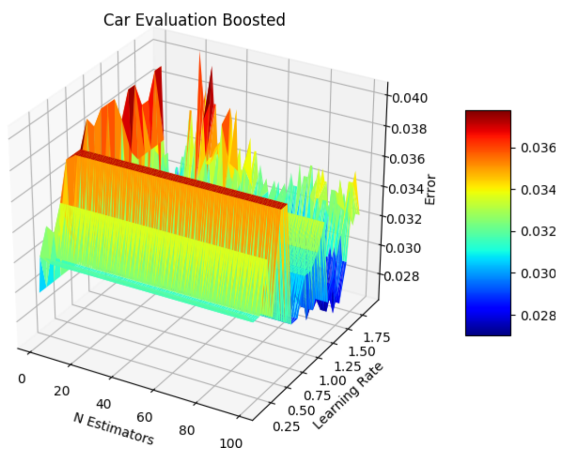

Figure 5: Car Evaluation Boosted

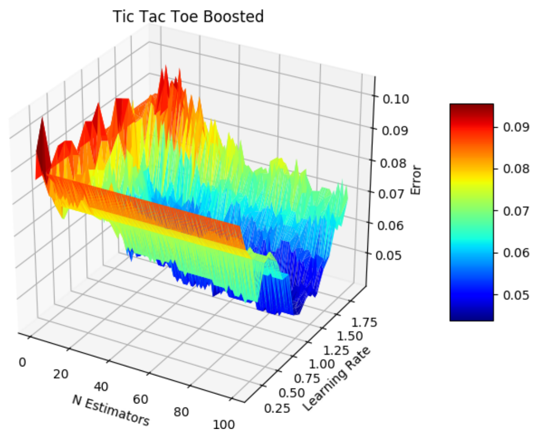

Figure 6: Tic Tac Toe Boosted

The first hyperparameter that was varied was the number of estimators. The number of estimators represents the maximum number of estimators at which the boosting is terminated. This parameter performs similarly as the maximum number of iterations in the neural net algorithm. Naturally, based on the function of this parameter, the error rate decreased as the maximum number of estimators increased because it allowed the classifier to relearn the incorrectly classified instances more times. The other hyperparameter that was varied was the learning rate. The learning rate represents the contribution of each classifier to the overall solution. A learning rate that is less than 1 means that each classifier's contribution increases, whereas a learning rate that is greater than 1 means that each classifier's contribution will decrease. For the car evaluation dataset, the optimal learning rate was around 1.4 and 1.6, and in the tic-tac-toe dataset, the optimal learning rate was around 1.4. With this in mind, the optimal learning rate in both instances demonstrate that for these data, this algorithm performs better by giving each of the estimators a lower contribution to the overall solution of the data, so the model is not designed to trust the previous estimators when classifying new instances of data.

The boosted algorithm trained quickly compared to the neural net algorithm; however, it trained slower than the decision tree algorithm, but this is expected because the boosted algorithm trains multiple decision trees in order to come up with its model. For both datasets, the boosted model did perform better than the previous two algorithms as it showed lower error rates for the optimal hyperparameters in this algorithm.

Support Vector Machines

For support vector machines, I implemented two variations of the *SVC* algorithm from *scikit-learn*. The SVC algorithm is a C-Support Vector Classification algorithm that is based on libsvm. To produce the two different variations of the SVC algorithms, I used two different kernels, poly and RBF. For both of these algorithms, they use a one versus one decision function for classification, meaning in multi-classification problems, each class is compared against another class resulting in ${}_nC_2$ comparisons. Below, I have presented the graphs from the poly kernel demonstrating the varying hyperparameters' impact on the error rate from *GridSearchCV*.

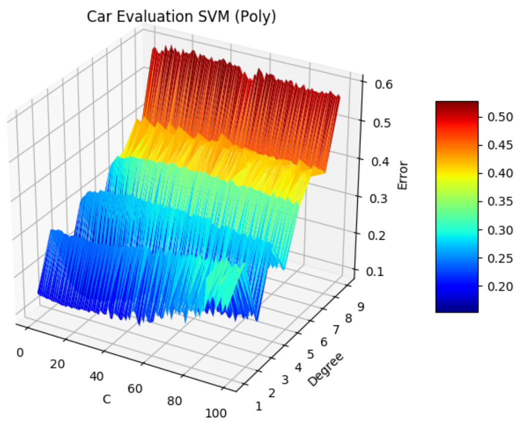

Figure 7: Car Evaluation SVM (Poly)

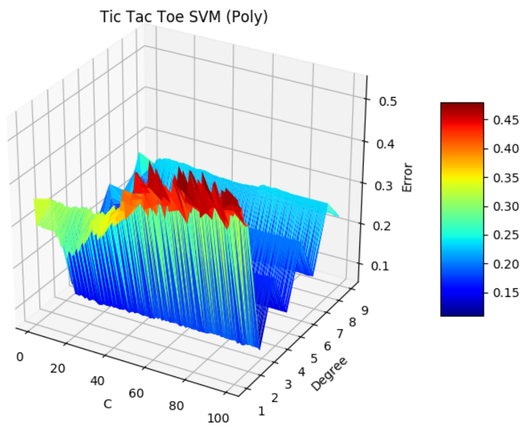

Figure 8: Tic Tac Toe SVM (Poly)

For the poly kernel, the SVM uses a polynomial to create a decision boundary for the data. For the tic-tac-toe data, the optimal degree is 3. As the degree increases from 3, the model begins to overfit the data resulting in a higher error rate. Once the degree passed 7, the model demonstrated a strong overfitting as the slope of the hyperplane increased drastically. For the car evaluation dataset, the optimal degree of the polynomial is 3 as well. Similar to the car evaluation data, the model began to slightly overfit the data as the degree increased from 3.

When comparing the poly kernel SVM to the previous algorithms, it can be seen that it did not perform as well as the decision tree, the neural net, or the boosted decision tree on either of the datasets. Additionally, the poly SVM took a longer time to train than the neural net, which demonstrates the idea that the SVM might not be the best model for either of these datasets.

Below, I have presented the graphs from the RBF kernel.

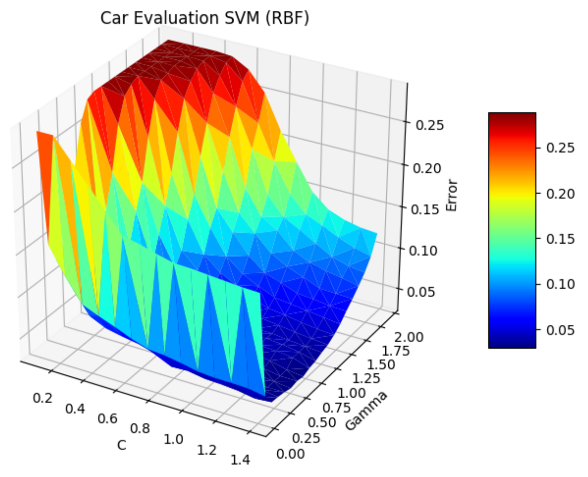

Figure 9: Car Evaluation SVM (RBF)

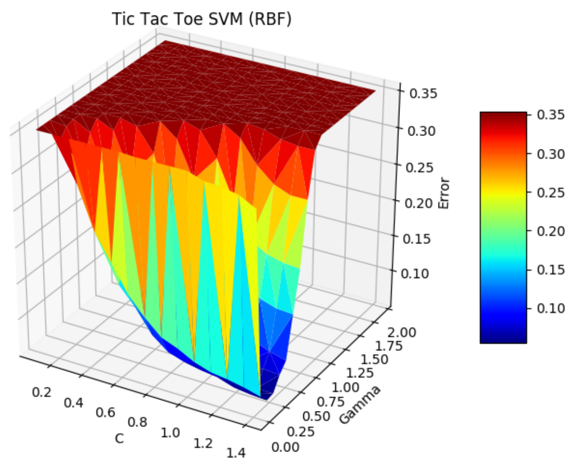

Figure 10: Tic Tac Toe SVM (RBF)

The RBF kernel stands for a radial based function kernel. The RBF kernel works by creating a Gaussian function around each instance of data to determine how similar instances are to each other as a way to classify the data points. The first hyperparameter that was modified was Gamma. Gamma represents the kernel coefficient for the RBF kernel, which means that this value determines how much influence each data point has in the instance space. A larger gamma value means that its influence stays relatively close to itself, whereas a lower gamma means that a data point's sphere of influence is a much larger distance. In the car evaluation dataset, the optimal gamma is between 0.5 and 0.75, which means that for the more accurate model, each instance of data holds a larger sphere of influence on the model. As gamma decreases from 0.5, the increased sphere of influence begins to result in a strong overfitting of the data, which makes sense given how more points would begin to influence classifications despite their large distance. Similar to the car evaluation data, the tic-tac-toe dataset also showed a low optimal gamma value, which was around 0.25. When gamma was increased or decreased from this value, the model showed signs of strong overfitting of the data because of similar reasons to the car evaluation dataset.

The second hyperparameter in both the poly and RBF kernel is the C-value. C represents the complexity of the decision boundary between 2 classifications. A lower C-value means that the decision boundary will prioritize larger margins, whereas a higher C-value will prioritize correct classifications. With this in mind, a larger margin for a decision boundary means a more generalized SVM classifier, whereas a smaller margin means a more specific SVM classifier that is focused more on classifying the data. With the poly kernel for both datasets, it can be seen that an increase in the C-value resulted in an increasing error rate, which can be attributed to overfitting of the data due to more focus on classifying the data presented. This means that the SVM began to trust the data too much as the C-value was increased. However, in the RBF classifier, the increased C-value seemed to have a decrease in error rate, but this could potentially have negative effects when comparing against the test datasets.

Between the RBF and poly kernels, for the car evaluation dataset, the RBF showed a smaller error rate than the poly kernel, but in the tic-tac-toe dataset, the RBF and poly kernels showed similar error rates for the optimal model. The RBF and poly models took a comparable amount of time to train, and considering how their error rates were no better than some of the previous algorithms, this shows that using SVM's for these datasets might not be worth the complexity and time tradeoff.

K-Nearest Neighbors

The algorithm I implemented for K-Nearest Neighbors was the *scikit-learn KNeighborsClassifier*. This classifier essentially uses a polling system, where it selects the "kclosest" neighbors to an instance of data to classify the new instance. Closest neighbors can be defined as points that are most similar to the new instance being classified. Below, I have presented the graphs of the varied hyperparameters for this classifier when using *GridSearchCV*.

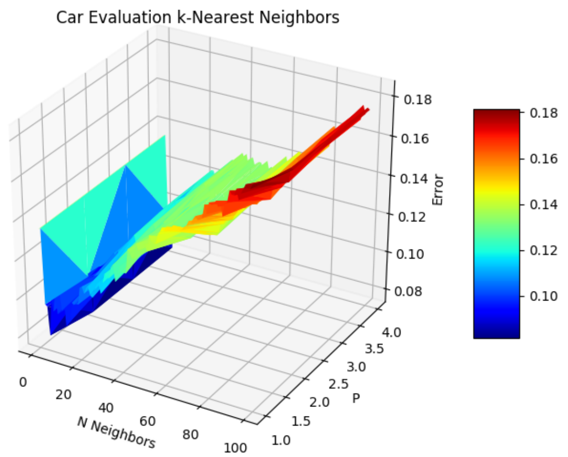

Figure 11: Car Evaluation Nearest Neighbors

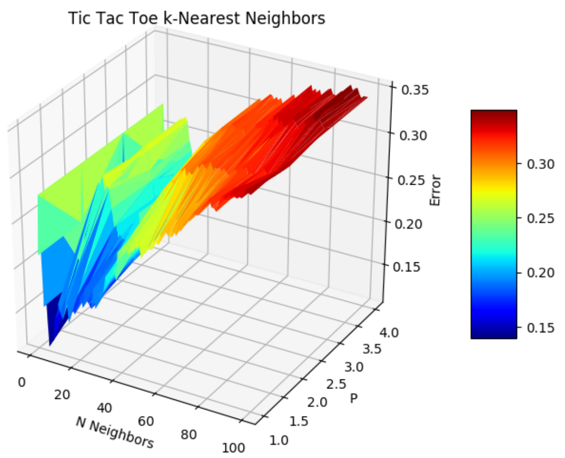

Figure 12: Tic Tac Toe Nearest Neighbors

The first hyperparameter that was varied was the number of neighbors used in the polling for the classification. For the car evaluation dataset, the optimal number of neighbors for polling is 5. Once the number of neighbors increases after this, the model demonstrates signs of strong overfitting as the error rate increases rapidly. For the tic-tac-toe dataset, the optimal number of neighbors is also 5. Like the car evaluation dataset, once the number of neighbors increased past 5, the graph shows signs of strong overfitting because of the slope of the hyperplane in the graph. This means that for both datasets, a more closely tied decision boundary results in better performance of the models. The other hyperparameter varied for the nearest neighbors' algorithm was the P value. P represents the power value of the Minkowski distance. A P-value of 1 represents Manhattan distance, a P-value of 2 represents Euclidian distance, and any arbitrary P-value greater than 2 represents an arbitrary power of the Minkowski distance metric. The tic-tac-toe dataset showed the best performance at a P-value of 1, whereas the car evaluation dataset performed best at a P-value of 4. This is not what would have been predicted because the tic-tac-toe data represents a higher dimensionality than the car evaluation data, so the P-value of the tic-tac-toe data was expected to be larger; however, the models show otherwise.

The K-Nearest Neighbors model trained very quickly, which is expected given that it is essentially performing a "lazy loading" classification when it is provided with test data, meaning it assigns classifications as it is provided with test data. When comparing this to previous models, it performed worse on the tic-tac-toe dataset, but its performance on the car evaluation dataset was very comparable to the other algorithms presented.

Learning Curves

The learning curves were created from the optimal models for each of the different algorithms found during the hyperparameter tuning discussed in previous sections. The x-axis represents the percentage of the training set that is used to train the optimal model from 10% to 100% of the training set. In the car evaluation classification problem, there are 4 potential classifications, which means that a random guess represents a 25% chance. In the tic-tac-toe evaluation classification problem, there are 2 potential classifications, which leads to a 50% accuracy for random guesses. With this in mind, the trained models performed significantly better than random chance. Below, I have presented the learning curves for the car evaluation dataset.

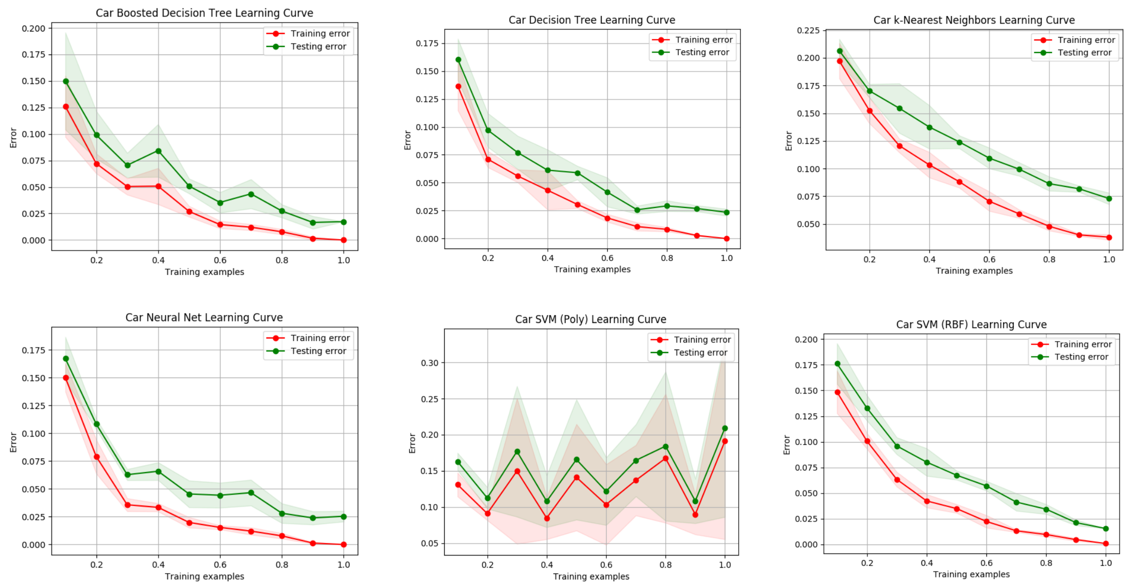

Figure 13: Learning Curves for Car Evaluation Data

In all of the above models, the testing error was higher than the train error. This was expected, since the model had not been exposed to the testing set until these graphs were generated. With regards to the shape of the curves, in almost all of the models, as the training size percentage increased, the testing and training error decreased with the exception of the SVM with the polynomial kernel. In that model, there is not exactly a clear trend as to how train and test error related to the increased training size. It has a trend of slightly increased error overall as the training size increases to 100%. In the neural net model, as the training size increases from 90% to 100%, the testing error starts to increase slightly. This is likely due to the model overfitting the training data considering how the neural net has 0 error on the training set at 100%. Overall, I would use the boosted decision tree model for this dataset. When comparing its error rate to other models when looking at training the model on 100% of the training set, it has one of the lowest error rates. The SVM with an RBF kernel has a slightly lower error rate;

however, the difference in error between the 2 models is less than 1%. This is a difference is a tradeoff I would be willing to make considering the large difference in training times and complexity. The boosted decision tree trains much faster and can classify new instances of data much quicker than the SVM, making it a more valuable decision. In addition, more data for this algorithm would result in even lower error rates because the boosting would continue to refine its classifications by focusing on the incorrect classifications to train its model.

Below, I have presented the learning curves for the tic-tac-toe dataset.

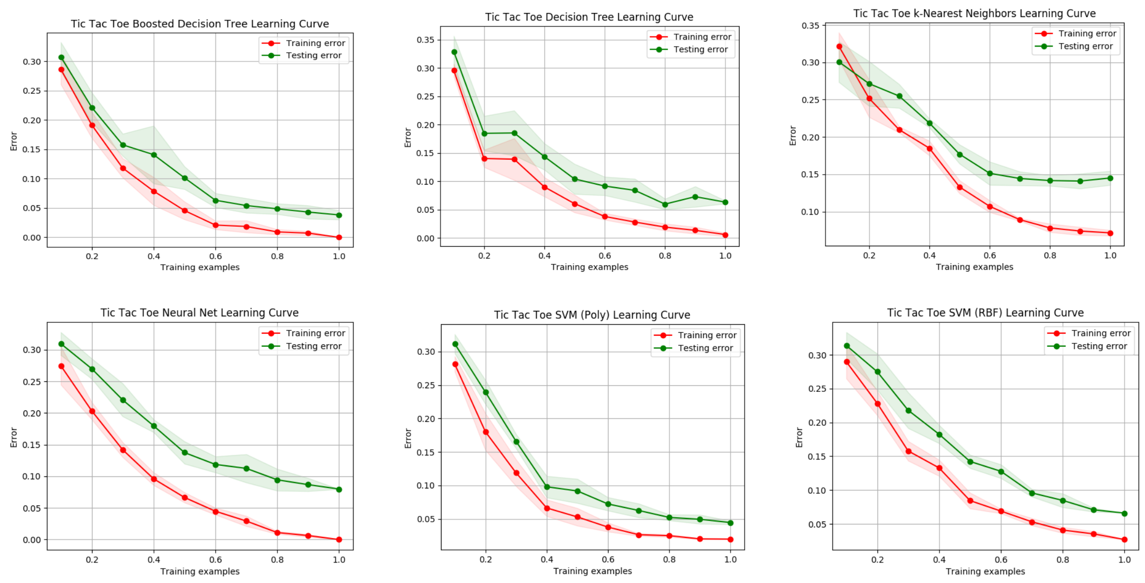

Figure 14: Learning Curves for Tic Tac Toe Data

In the tic-tac-toe dataset, in each of the models, the train error was lower than the test error, which was expected for the same reasons as the car evaluation dataset. Similar to the car evaluation dataset, for each of these models, as the percentage of training examples trained increased, the testing error and training error decreased as well except for a few cases. In the decision tree model and the k-nearest neighbors' algorithm, after the number of training examples increased past 80%, the testing error began increasing. This is likely due to the model overfitting the data when being trained on a higher percentage of the training set, which is why the test error began to increase while the train error decreased. Similar to the car evaluation dataset, I would also utilize the boosted decision tree model for the tic-tac-toe dataset as well. When trained on 100% of the training set, it presented with the lowest error rate. The neural net model and both SVM models showed comparable error rates; however, due to their complexity and lengthy training time, they would be overkill for this data, since the boosted decision tree model performed better with a less complex data structure and less train time.

Conclusion

These two classification algorithms were very valuable in demonstrating key differences and similarities between some of the most common supervised learning algorithms. The usage of the 5-fold cross validation with *GridSearchCV* from *scikit-learn* tuned the hyperparameters of each of the models to create the best model to test. The learning curves created from testing the test set on the best models provided valuable information into how each of these models would truly perform on unknown instances of data. Overall, these learning curves show that the boosted decision trees models performed best on the car evaluation and the tic-tac-toe datasets.

References

Source of Algorithms: Python3 scikit-learn package - https://scikit-learn.org/stable/modules/classes.html
Car Evaluation Data: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
Tic Tac Toe Data: https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame
Code: https://github.com/tobincolby/CS4641-SupervisedLearning