Nilay Mehta

CS 4641 – Machine Learning

2/4/2018

Supervised Learning

Introduction: The purpose of this paper is to outline five different techniques within supervised learning and the nuances of these algorithms when trying to classify two classification problems. The structure of this paper will be an introduction to the classification problems, algorithms and implementation details, and overall analysis for each algorithm. All the data and graphs found in this paper can be simulated by running the instructions to the experiments in the readme file. The two classification problems I chose to analyze are:

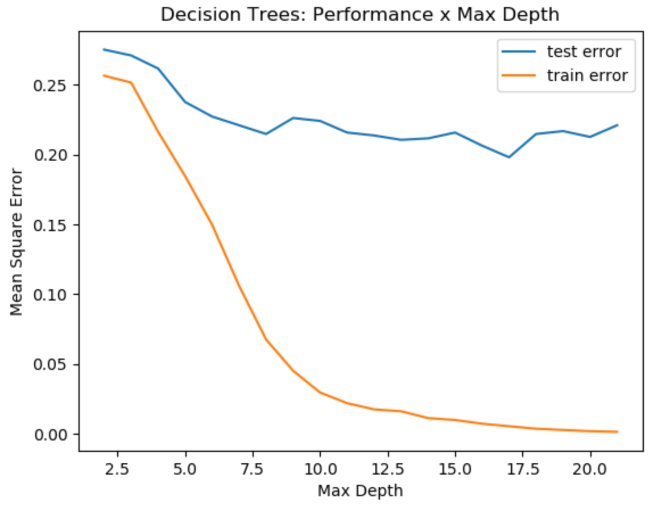
- **Human Resources Analytics**: This dataset explores had a variety of characteristics targeted to explain why employees leave prematurely from their occupation. It proposes the classification problem of identifying potential employees that hare likely to leave next through a series of attributes, some of which are the employee's last evaluation, average monthly hours, history and time spent with the company, the department they work in, and salary range. There is a grand total of 14,999 data entries with 10 unique attributes each. This dataset was intriguing to me because of its application in the real world and potential complexity in classification. I also plotted the distribution of the attributes and, because many of them have multiple classifications and the frequency and distribution of attributes also varies significantly. In the code, this dataset is referenced as 'HR_Analytics' or 'HR' and was made available under a Creative Commons, CC By Attribution-ShareAlike 4.0 License from Kaggle.
- **Handwriting Verification**: A dataset that contains information captured by a system that tracks gestures and movements while someone is writing. The dataset consists of 3197 individual samples of data with 52 attributes each. The last column in the dataset contains the label which denotes if the handwriting is unique to one person or contains characters that are mixed between multiple participants. It can be modeled into a classification problem to try and predict the degree that the handwriting samples match, and if they differ, to denote multiple authors to that data point. This dataset was appealing to be because it seemed like a unique classification problem from the datasets I narrowed my search down to. I was also interested to balancing my other dataset with a classification problem that had a lower sample size, yet significantly higher attribute count. In the code, this dataset is referenced as HV and was made available under a Creative Commons, CC0: Public Domain License from Kaggle.

Further information, including the original linked to each dataset, can be found in the readme file. All the algorithms and analysis have been run on these two datasets and the data was partitioned following a 70/30 ratio for training and testing sets respectively. Each partitioned set was shuffled and sampled randomly and follows the guidelines discussed in class. For runs that varied the training sample size, the testing sample set was still maintained at its original 30% size. Each algorithm was run multiple times with nuanced hyperparameters to determine the effects of each of these. The overall purpose of these tasks was to find the hyperparameters that optimized the accuracy while observing training characteristics like over/underfitting. The metric used to evaluate the classifications in both the training and testing sets was the Mean Square Error.

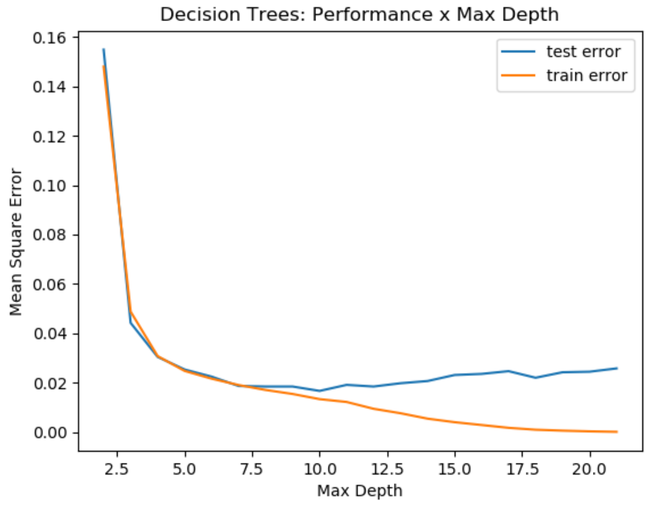
Decision Trees with Pruning

The Decision Tree package I used was SciKit-Learn's Decision Tree Classifier and the pruning parameter was the max_depth for the tree. The reason for implementing pruning is to minimize the complexity of the overall tree and this reduce overfitting. During my experiments, I assessed the performance of the decision trees based a few parameters such as accuracy over different max depths and accuracy over different training samples with a consistent max depth.

HR Analytics



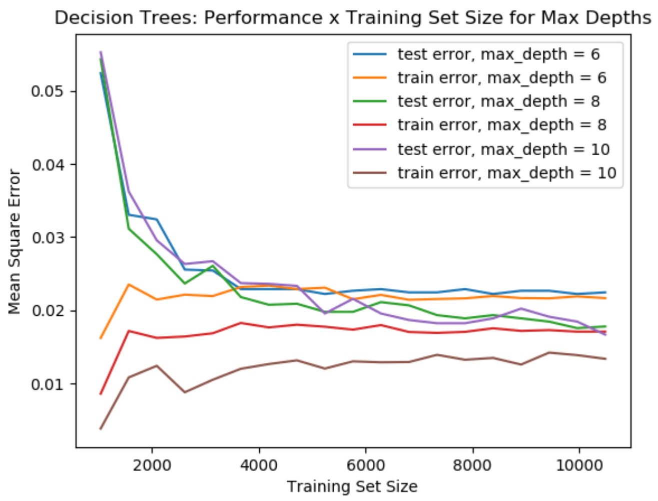
Handwriting Verification



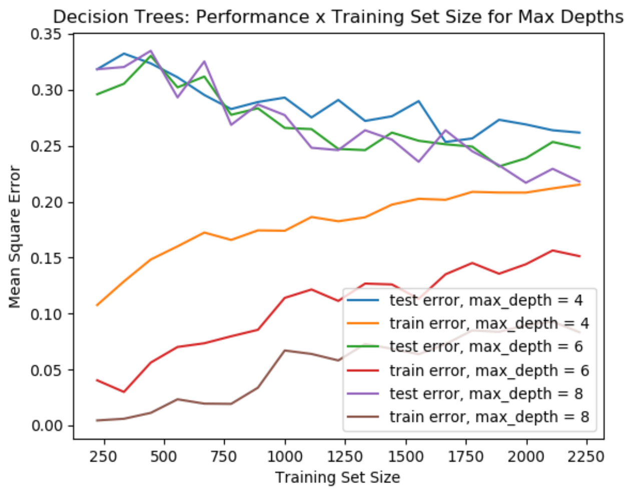
When I tested the performance of different max depths of the HR analytics dataset, the train error continued to diminish until it was negligible. While this behavior is not surprising as we expect the training error to continue to decrease as the depth of the tree increases, it was interesting to note that there seems to be an inflection point around a max depth of 10, which is also the number of attributes that this dataset contains. Another observation is that the test error continued to stay relatively stable suggesting that the model began overfitting on the data as the depth increased. For the handwriting verification dataset, we can see that the initial error for both the train and test error starts out relatively low, but continues to dramatically decrease until it starts to level out after a max depth of 5. One possible explanation for this is that, in this dataset, there are a dew attributes that carry higher weights than the rest, and are much more instrumental in determining the final outcome of verifying if one or multiple people contributed to the written sample. It's also interesting to note that after reaching a max depth of 7, the test error starts in increase slightly while the train error continues to diminish, again also suggesting that the model is overfitting the data.

The distributions for the datasets was significantly different, which was likely the main contributer to the depth needed to optimize error for the datasets. For the HR analytics dataset, nearly 76% of the data given supported one classification alone, while the handwriting analysis dataset was much more balanced with a 52/48 ratio between classifications. Because the handwriting analysis has a much more balanced representation of classes, it had a significantly lower error and lower optimal max depths reletive to preformance.

HR Analytics



Handwriting Verification



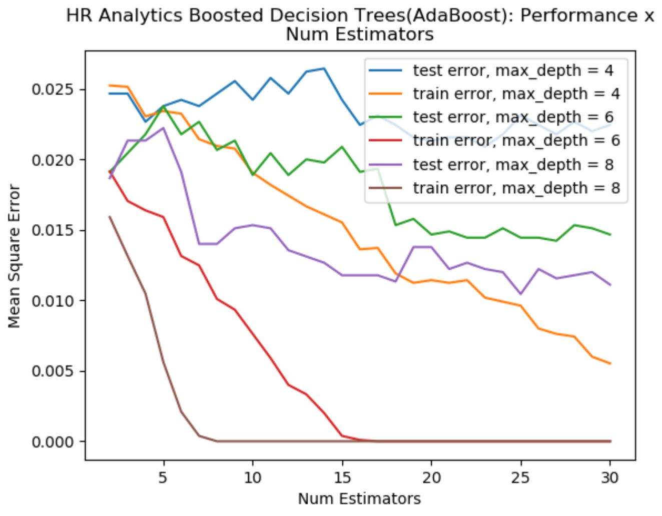
After determining the relationship between max depth and performance and also observing potential inflection points in the graph that cause overfitting, I moved on to exploring the relationship between train and test errors at constant depths for the tree (graphs can be found on the following page). For the HR Analytics dataset, I chose to center my observations around the depths 6, 8, and 10 and found that a sample of 3500 data points was sufficient across all depths to produce a decision tree with very low error. However, the mean square values overall were very low, indicating exceptional performance across all of these trees. The depths for the handwriting verification dataset I chose to focus on were 4, 6, and 8. Like I mentioned previously, despite being the dataset with significantly more attributes than HR analytics, it seems like the handwriting verification dataset has attributes with noticeably higher weights in influencing the classification. Like the other dataset, the handwriting verification dataset has a consistent test error, but the train error seems to increase across all depths as the training set size increases. It seems to converge close to a mean square error value of 0.20, which means that as the training set size increases, there is an overall increase in train set error. One possible explanation for this is that as the training set gets larger, we overfit the training set less because of the pruning (max depth limitations). As a result, the training error continues to rise as the training set increases.

Boosting(ADABoost)

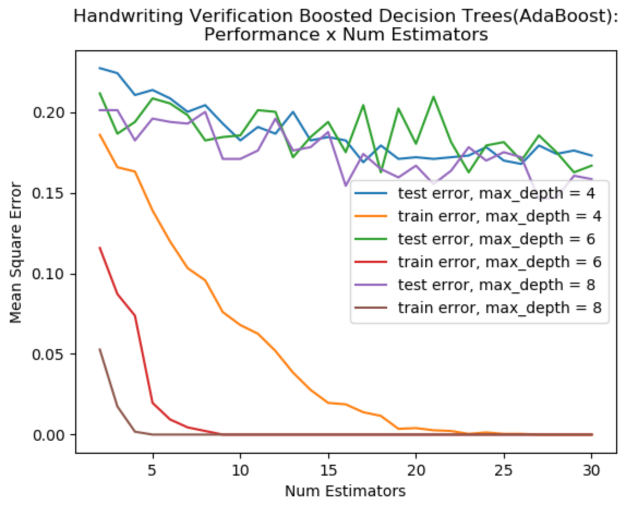
Adaptive boosting, or ADABoosting, is an ensemble supervised learning method where you prepare a weak classifier and continue to add to it sequentially by training it using weighted data. Because incorrect data is weighed higher, it continues to learn and focus on the incorrect data and ultimately becomes a stronger classifier. The variable that I manipulated was the number of estimators across multiple max depths.

My main comparison was the performance of the tree based on the number of estimators used while boosting. While the values varied based on the max depths of the trees, for the handwriting verification dataset, as the depth increased, the number of estimators added had little effect on the train data but continued to significantly decrease the test error, suggesting that the model continued to learn while avoiding overfitting (because test error continued to decrease despite stagnating train error). For the handwriting verification dataset, across all train errors, the increased number of estimators had little effect on the overall performance of the classification. This

<u>HR Analytics</u>



<u>Handwriting Verification</u>



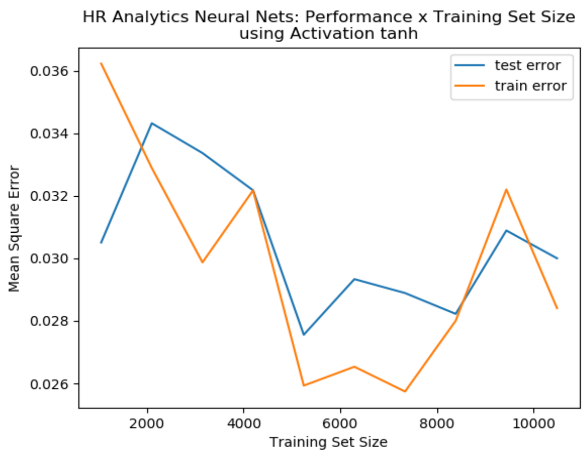
Using the observations from the previous experiments, I decided to prune the initial decision tree classifier across the depths 4, 6, and 8. For the HR Analytics dataset, at a depth of 4 and 6 (although the effects on 6 are minimal compared to 4), as the number of estimators increased, the accuracy continued to increase. This paired with the reduction of more than half of the initial error indicates that the boosted versions of the decision trees does learn and classify the data better. However, the algorithm is also successful in increasing accuracy of predictions, but the training errors decreased which raises the concern that the model might be overfitting to the data. But, the error values are significantly lower and the difference is minimal relative to the performance of the pruned, non-boosted tree at max_depth = 4.

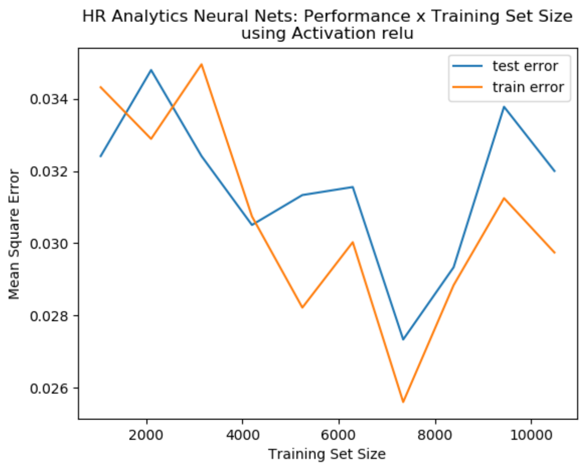
For the Handwriting Verification dataset, we can see that these effects are even more apparent; the test error maintains a tight threshold across the various max depths and estimators. This means that this algorithm was not overfitting to the data. We can make this assertion that it was not overfitting because test error was relatively stable. This paired with the decreased training error suggests that ADAboost using a Decision Tree classifier was more successful on this dataset than the HR Analytics dataset when considering improvement from the non-boosted, pruned tree performance.

Neural Networks

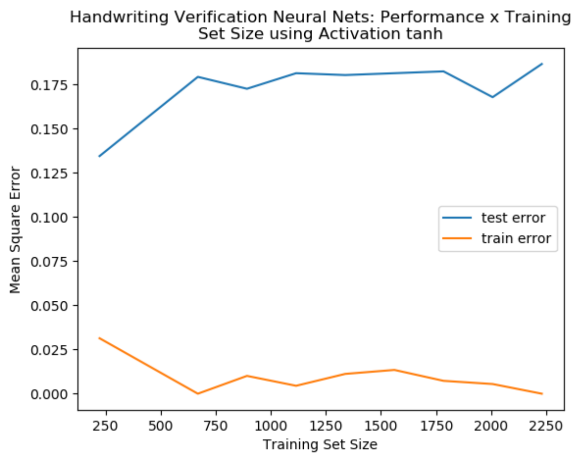
The classification I used for my neural network was Scikit-Learn's multilayer perceptron(MLP). My results for neural networks were probably some of the most surprising in my experiments. The expectation was that it would be able to perform on par with the other algorithms discussed in this paper, but even with the default parameters, it was very accurate wile having minimal training times. Even after tweaking multiple parameters and rerunning the algorithm more times than I could count, the performance increases were minimal, but still impressive compared to the accuracies of the previous algorithms. The main hyperparameters that I tweaked were the hidden layers (neuron count and layer count), max iterations, and activation functions. There were too many graphs with varying performances, but maximum performance I was able to achieve through neural networks was a mean square error of 0.026 at training batch sizes of around 7000 for the HR Analytics dataset. However, for the handwriting verification dataset, this algorithm performed better on training data and vastly superior when cross validated, yielding a mean square error better than any of the other algorithms (graph on following page). One explanation I had for this was, because of the dataset's complexity and high attribute count, it was able to deliver more accurate predictions because I used a higher hidden layer count (3 hidden layers), ranging from 22 to 10 neurons in each layer. Across both datasets, when tweaking the hidden layers there was a diminishing return with adding more layers and more neurons in each hidden layer, while the training time increased a significant amount. For both datasets, when adding four or five hidden layers opposed to the three hidden layers I finalized on, increased the accuracy a negligible amount. Neuron count had the same effect. The two configurations I tested for neuron count within the hidden layers were uniform count across all hidden layers, and a tapering count where the first hidden layer had the most neurons, and the

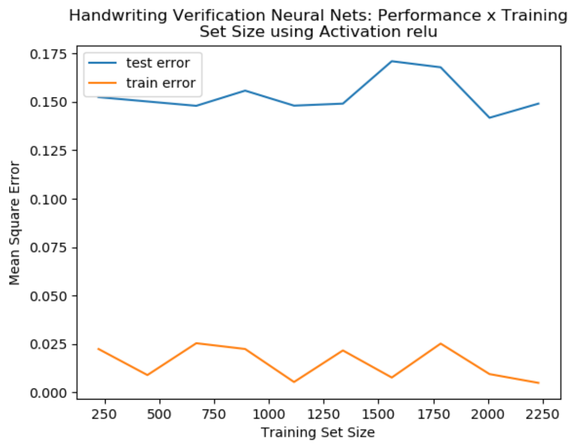
HR Analytics





Handwriting Verification





last hidden layer had the least. I found that the uniform neuron layout worked better for the HR Analytics dataset and while the tapering neuron layout worked better for the handwriting verification dataset. One possible explanation for this is that the increase in hidden layers influences the generalizability of predictions. But this creates issues when the network becomes too complex for the size of the dataset, especially with a low iteration count.

This experiment's main takeaway seemed to indicate that the fewer the layers and neurons, training time would be impacted in a positive manner. With respect to neuron count, reducing the number of neurons as indicated with the tapering configuration allows each layer to learn features and prioritize which features have the greatest impact, ultimately allowing the network to produce the correct classification.

With regard to the activation functions, the performance for both datasets was generally better for the tanh activation function, although the effects were almost negligible. According to resources online, the tanh activation function outputs a range of outputs using sigmoidal truncated between positive and negative 1. Because most inputs that are more negative or positive will map to their respective outputs, a tight range of values will result in near-zero values; this means that overall network performance will increase because it is less likely to be bottlenecked with these low values.

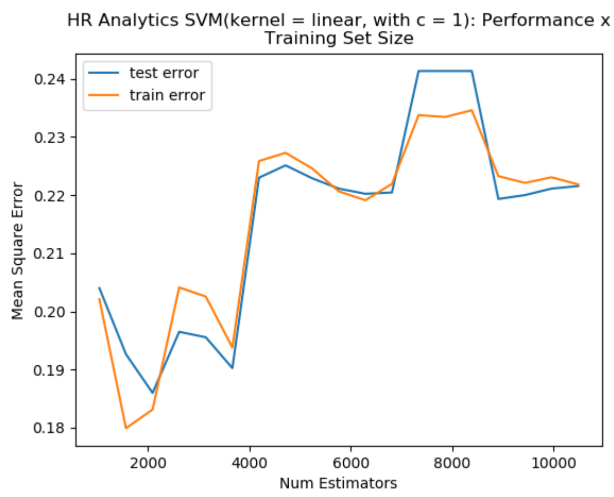
A reason this algorithm's performance was so scattered for HR Analytics could be explained by the performance of the neural network over different training set sizes. The jittery shape of the graph for the HR analytics dataset on the previous page suggests that the model may suffer from the curse of dimensionality. This means that the neural network is not able to decide which features are more important because it doesn't have enough training data.

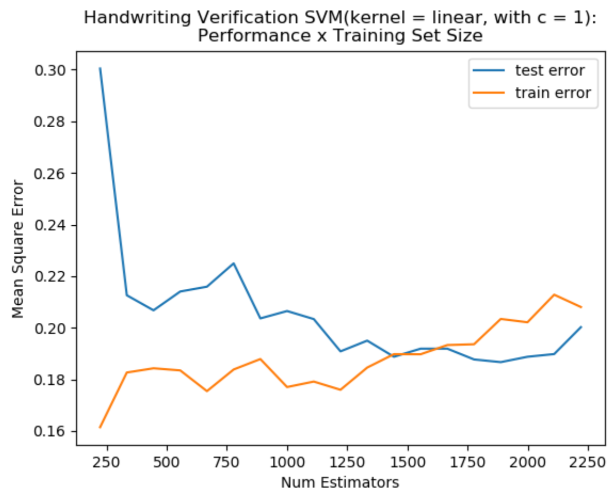
Support Vector Machines

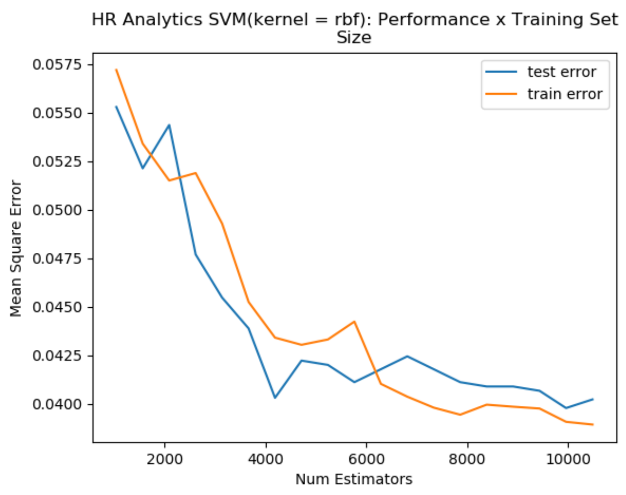
Support Vector Machines work by categorizing the data passed in by finding the parameters that work best for the chosen kernel. It uses the function and compass various mappings of the data to classes; the support vector machine was implemented using Scikit-Learn's SVC, with a focus on the two kernels: linear and radial bias function(rbf).

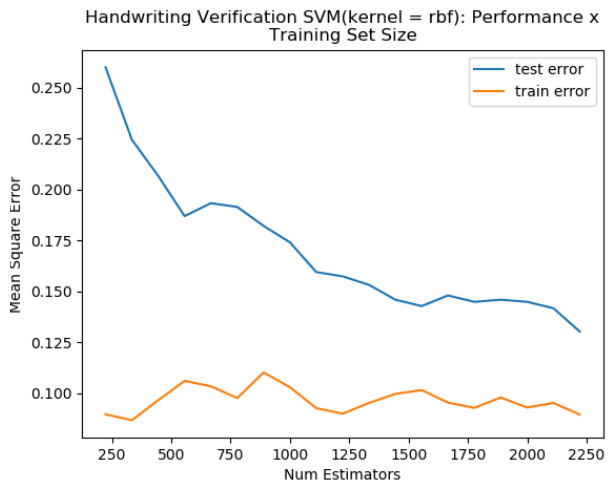
HR Analytics

Handwriting Verification









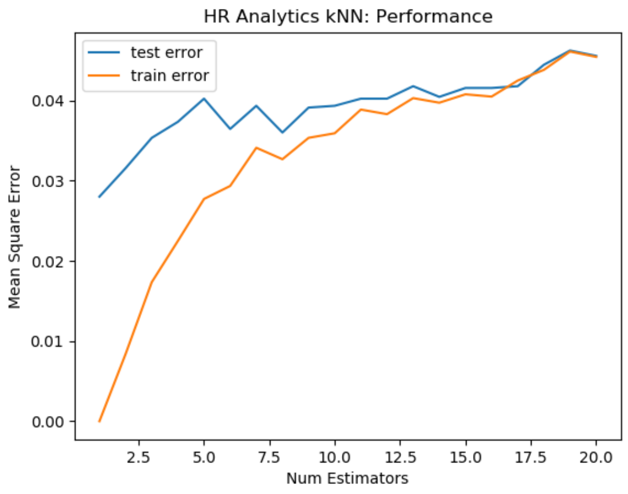
Note: the x-axis along all of these graphs should be labeled “Training Set Size”, not “Num Estimators”.

These experiments took the longest time to train out of the batch, but the model seems to have a relatively high accuracy. I was most impressed with the level of precision that was accomplished for the HR Analytics dataset. The train and test errors seemed to be almost identical, although it was strange that the linear kernel saw performance degradation as the sample sizes increased for that dataset. However, for the other dataset, both kernels seemed to preform adequately, while the rbf kernel did slightly outperform the linear kernel as the training set size increased.

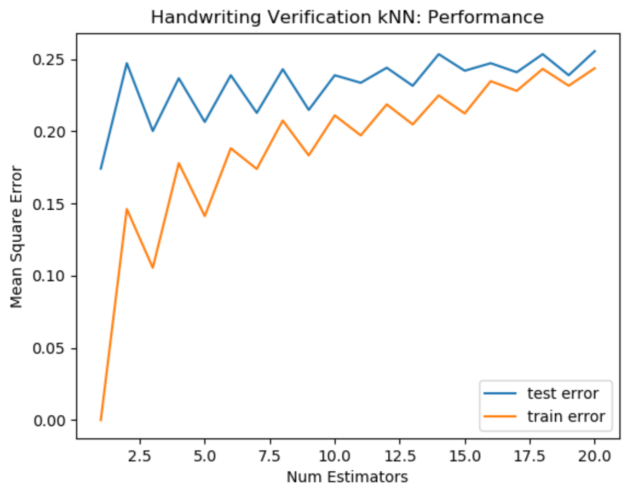
I was surprised that the rbf kernel was able to deliver a classification accuracy that surpasses any of the other algorithms for the handwriting verification dataset. Even thought this was by far the longest model to train, the accuracy payoff seemed to be worth it. Although it is not shown above or in any graphs in this section, I did ty tweaking certain parameters in the linear function like c. The default value of c is 1, but increasing this value leads to overfitting. The train error continues to diminish to 0 while the test error stays relatively stable across most values of the hyperparameter.

k-Nearest-Neighbors

HR Analytics



Handwriting Verification



The last algorithm in the set was k-Nearest-Neighbor and was implemented using the k-neighbors classifier in scikit-learn. The parameter I tweaked was the k as the number of estimators and used to group the data and output the final classification and I plotted this across its error over the k values [1, 20] inclusive. The highest accuracy for both of these datasets occurred when k = 1 and the overall trend for the datasets was a negative effect on both the train and test error, while plateauing out after k = 6 for the HR analytics dataset and k = 3 for the handwriting analysis dataset. Conceptually, this makes sense because as you increase the number of k, which is the number of neighbors used to make the prediction, the accuracy of the result logically decreases. The performance of this algorithm still had more test error than other methods discussed in this paper and one reason for this is that the data itself is not a good fit for this algorithm. kNN thrives on data that is closely clustered, locality, and smoothness over dimensions. These datasets did not follow these characteristics, and thus the recommended k will be lower.

The handwriting verification dataset did have an interesting accuracy plot with multiple spikes up and down as k increased. The reason for this is most likely the dimensionality of the model across attributes. The data is grouped in odd pairs, at least in distance across dimensions, and increasing the k at odd intervals takes advantage of this characteristic, yielding an improved train and test score. The inverse is true at even intervals which contributes to the drop in accuracy.

Conclusion

Overall, each algorithm did a proficient job for classifying both datasets relatively accurately. As a whole, the HR Analytics dataset seemed to have a higher accuracy across all the algorithms when looking at test error.

For the HR Analytics dataset, the algorithm that had the best overall test error was the boosted decision tree using ADABoost with a reported average MSE test error of 0.016, with the neural network coming in as second (reported average MSE test error of 0.031). Despite this dataset being real world data that was collected in larger corporate cultures across the United States, it was surprising that the data was able to consistently deliver a high accuracy of classification on all the algorithms. After these algorithms, k-nearest neighbors came in with the third best accuracy (average MSE test error of 0.04), followed by the SVM running the rbf kernel (average MSE test error of 0.046). The algorithm that preformed the most poorly on this dataset was the pruned decision tree with an average MSE test error of 0.233. It was surprising to see just how significant boosting improved the decision tree's performance, a trait that was not present in the other classification problem. Even when factoring in time spent to train the models, the algorithm's rank is still consistent relative to their accuracy score. For the Handwriting Analytics dataset, the algorithm that resulted in the lowest test error was the neural network with an average MSE test error of 0.124. The SVM classifier was a close second with an average MSE test error 0.151. The rest of the algorithms with their respective MSE test errors are: k-NN as third with an error of 0.177, the boosted decision tree with an average MSE test error of 0.185, and last the pruned decision tree with a MSE test error of 0.263. In retrospect, it is interesting to see how the two datasets and the optimal algorithms compare. Simpler, more conventional methods of classification like pruned decision trees and boosted decision trees seem to thrive on datasets with a low number of attributes. Algorithms with generally higher training times and coincidentally hyperparameters to tweak like neural networks and SVMs perform better when there are a higher number of attributes; Lastly, kNN consistently preforms adequately, but is an expensive algorithm to run on large datasets like HR analytics. Out of the last three algorithms, it seems as though kNN is a powerful predicter when there was less data available, SVMs are incredibly accurate at generalizing results when there is sufficient data available, and neural networks are a compromise between the two and can perform consistently well, even under default hyperparameters.