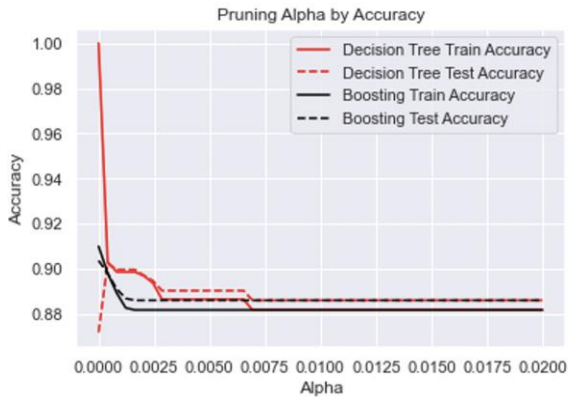
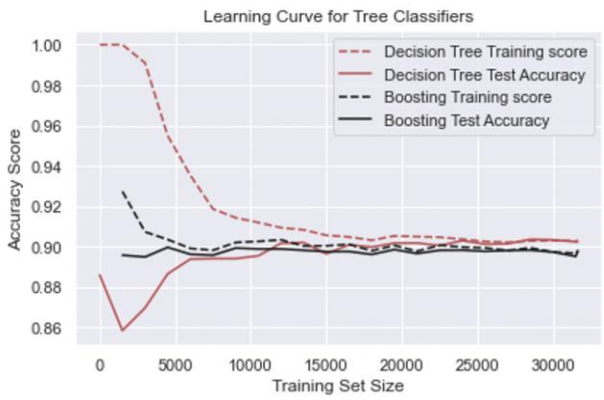
For this assignment, I chose the datasets for algorithm implementation based on dataset size, ability to work with multiple algorithms, and whether I found it an interesting problem to analyze. Dataset size was the biggest consideration. The dataset needed enough data to split into nontrivial training and testing sets. However, the dataset could not be too big because the assignment's analyses required looping and multiple runs with different parameters that could take a long time with my limited computing resources if the data was too large. Both datasets are binary classification problems that had a clearly defined target concept and features for target prediction. Additionally, these datasets fit well with the code examples I found. As for finding the problem interesting to analyze, I chose the job churn data for data scientists because I work as a data scientist and am considering some job change so the dataset seemed a fitting choice. I chose the bank marketing data because I work in the financial industry and am interested in predicting things like whether a person will buy a financial product.

First Dataset

For the first dataset, the target concept is whether a customer will buy a financial product. The first dataset. Features include demographic information such as age and marital status as well as some of the customer's history with the banks.

Decision Tree and Boosted Tree Classifier





In examining the pruning graph, for the decision tree see that with an alpha of 0 the tree is allowed to grow unbounded, and the testing accuracy is at its lowest and the training accuracy is at its highest. This is because the decision tree is allowed to overfit on the data. With no pruning, the tree will grow to have 5,115 nodes and be 30 layers deep. The tree with the best pruning measure, an alpha of .0004, only has 49 nodes and is 10 layers deep.

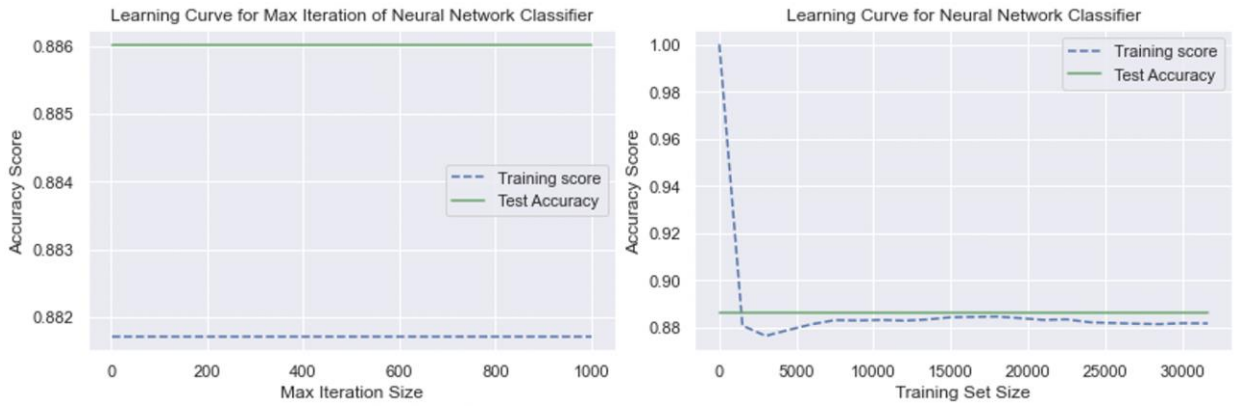
The gradient boosted tree algorithm is a combination of many decision tree regressors into one classifier. For the gradient boosted tree, the best preforming alpha was 0, which aligns with the expectation that more aggressive pruning produces similar performance to a regular decision tree with a boosted decision tree.

For both the regular tree and the boosted tree algorithms, as the alpha grows both the test and training accuracy decrease but level off after a certain point. After a certain alpha, both the decision tree and gradient boosting tree algorithm only grow the trees to 1 node hence the lower performance. The higher alpha makes the classifiers underfit the data.

For the decision tree's learning curve, as the training set grows, accuracy increases on the testing set and decreases on the training set. It levels off around 15000 and even decreases slightly at the end. This might happen because the training set with a smaller amount might not be completely representative of the whole distribution and the more information introduced, the

harder the actual prediction becomes. A similar pattern is followed for the gradient boosted tree classifier. Training set accuracy decreases, and testing set accuracy increases as the training set grows larger. The two sets of accuracies start off much closer even when the training set is a lot smaller. And the gradient boosting tree learning curve levels off much faster than the decision tree's.

Neural Networks



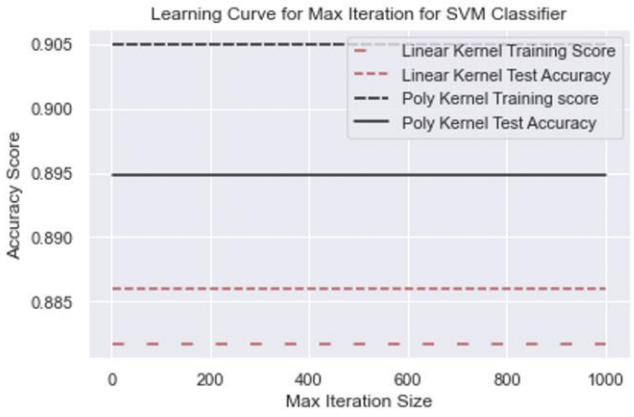
I first tried to use lbfgs, a quasi-Newton method for activation, as my activation function but it gave me a warning that it had not converged. Even with larger iterations like 200,000, that activation function still did not converge. It is possible that it would eventually converge at even larger iterations but that is an uncertain possibility, so I did not explore that. It was also interesting that even though it did not converge, the testing accuracy of the resulting classifier using lbfgs, .88, was not that much lower than other classifiers. The other activation function I tried and used to produce the graphs above was sgd or stochastic gradient descent. This activation function did converge.

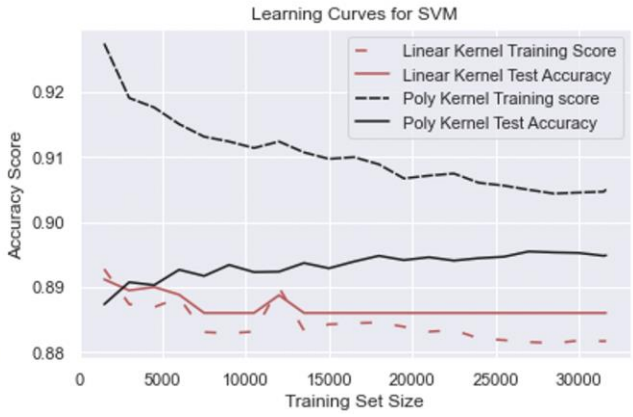
When creating the learning curve for the number of iterations, smaller iterations using sgd as my activation function gave me the same will not converge warning as using lbfgs did. Unlike lbfgs, with larger iterations the warning stopped. Interestingly, the accuracy was the same for all

number of iterations even when sgd did not converge. Training size decreases the accuracy of the neural network on the training data, but the accuracy remains the same on the test data. This might mean that the neural network creates the most accurate classifier with little data.

There were other things that I left static for this analysis that could have impacted accuracy using this implementation of a neural network. I could have changed the learning rate, the size of the hidden layers, or even the random state. Cross validation as implemented in python, I think would have been an effective way to systematically search through these parameters and compare which permutation gave the best results. I decided to leave those static and explore how iterations, different activation functions, and training set size impacted testing accuracy and error rates.

Support Vector Machines (SVM)





The learning curve comparing the number of iterations and accuracy shows that more iterations did not increase the accuracy of support vector machine for both poly (polynomial) and linear kernels. This may be because with both kernels, they converge right away and there is no need for the extra iterations to update the kernel and improve performance.

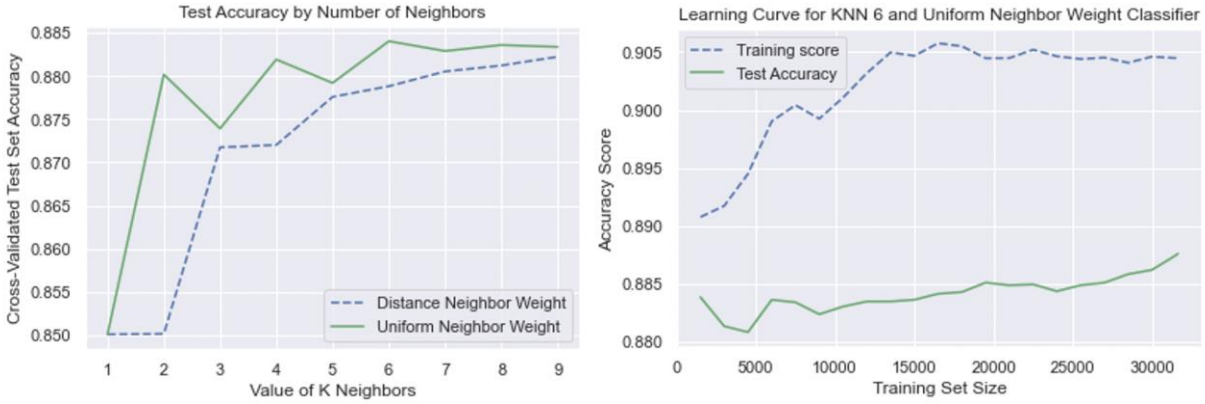
The learning curve for the size of the training set shows that for the linear kernel training and test sets for the linear kernel both decrease over time, although there is a spike between 10,000 and 15,000 training set size. This seems to imply that with less data, the classifier

performs better. This could be because of the way the training data subset was chosen. The subset then may not actually be the most representative sample of the whole dataset and may lead to higher performance than a representative sample of the data size would give.

For polynomial kernel, the training accuracy decreases as the testing set increases. More data is necessary for the best performance with this classifier. Another interesting input for the polynomial kernel is the polynomial degree. The default which I used is a degree 3 polynomial. Increasing that degree could have overfitted the training data and decreased the test accuracy or it could have improved the test accuracy. I did not explore that because I was mainly interested in the difference between kernels and not the iterations upon one kernel.

In both the iteration graph and the training set size graph, the accuracy of the poly kernel on both the training and test datasets was higher. This could be because the data is not actually linearly separable and the linear kernel, therefore does not create the best possible classifier for this dataset. The polynomial kernel is better able to capture the complexity of the dataset and therefore be a more accurate classifier.

K Nearest Neighbors



I used cross validation here to help me pick out the best parameter for weight of distance measure and number of neighbors. The cross validation here was helpful because it was able to

run multiple combinations of different model inputs and get the most accurate version of the algorithm. The graph for cross validation shows that more neighbors do not necessarily equate to higher accuracy. Six neighbors with a uniform weight for every neighbor had the highest accuracy.

In the graph for the learning curve, it shows that more data improved the accuracy of the classifier on both the test and training set. This makes sense because the more data there is the more likely the new data has closer datapoints to it. The closer data points would give better predictions.

Comparative Analysis

	Classifier Name	Training Error	Testing Error	Testing Accuracy	Training Time
0	K Nearest Neighbor	0.095712	0.112504	0.887496	0.006506
0	Support Vector Machine	0.094954	0.105205	0.894795	15.62847
0	Gradient Boosting Tree	0.090056	0.096358	0.903642	3.40101
0	Neural Network	0.118273	0.113978	0.886022	1.535159
0	Decision Tree	0.096976	0.097538	0.902462	0.173975

The accuracy of all the classifiers is a lot higher than I was expecting. I think part of that is that 88% majority of the data is a 0 response so even if you just guess 0 for everything, you could still get a classifier that predicted right 88% of the time. K nearest neighbor, support vector machines, and neural networks barely perform better than my bad call everything 0 classifier.

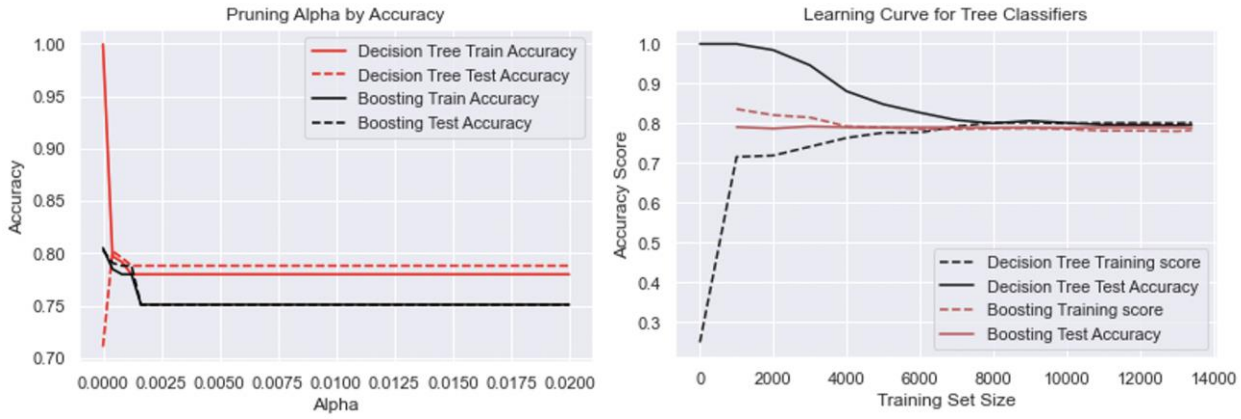
Since the testing error signals the classifier's ability to generalize well, gradient bosting tree is the best classifier in that regard. However, decision tree's testing error is not that much higher, and it also had the second fastest training time. If I had to choose the best algorithm, for its accuracy and training time alone, I would pick decision tree. It has the benefit of being a very explainable model for the kind of information a marketing department might want to glean from

this dataset, such as what kind of customers buy this financial product. Decision trees, implemented with python, give you visualizations, like the actual tree that shows what features are most important for grouping customers. It also gives you a feature importance calculation that also can show what features are not actually helping score new customers. The decision tree is accurate, fast, and understandable so it is the best algorithm for this dataset and problem.

Second Dataset

The second dataset's target concept is whether a student who took a course offered by a company wants to continue to work for the company after course completion. The predictors include demographic, education, and job experience.

Decision Tree and Boosted Tree Classifier



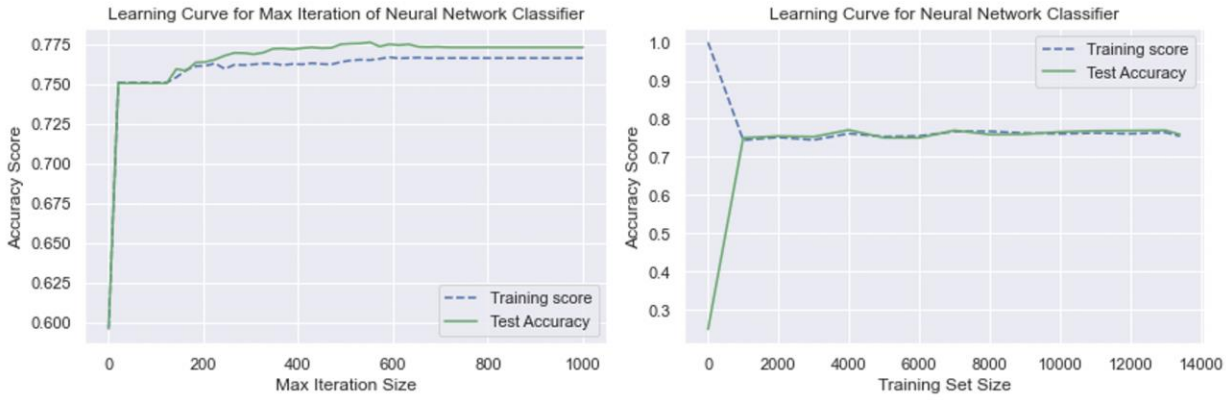
A decision tree trained on the data without any pruning resulted in a large tree with. 6,131 node that was 42 levels deep. With an alpha of .0004 the tree only grew to 35 nodes and 8 levels and had the highest accuracy score.

As the training set size increased, the accuracy increased for both predictions on the training and testing set. Accuracy starts to decrease around 4,000 rows in the dataset, it increases again around 8,000 and decreases around 11,000. I again suspect this has to do with the way I

chose the subset of data. For the gradient boosting tree, the accuracy on the testing with a small sample size has already levelled off at around 2,000 samples. The training accuracy decreases while the testing accuracy stays level.

I was again able to prune more aggressively the gradient boosting classifier. The most performant alpha was 0. With that alpha, the trees that made up that classifier only had nodes between 0 and 15 and were of max depth of 3.

Neural Networks

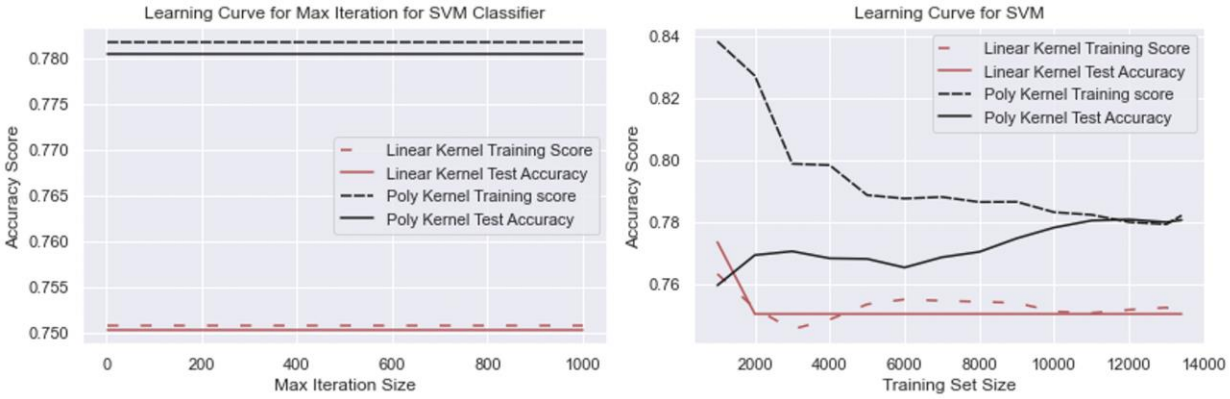


The activation function lbfgs did not converge for the hr data either. Sgd did converge. They both had similar accuracy scores .75 for lbfgs and .76 for sgd. For this problem, I wanted to understand if an equally performant classifier was still possible even if the activation function never converges.

Using the lbfgs, more iterations increased the accuracy of the classifier on both the training and testing sets. This makes sense because it is possible that I just have not iterated enough over the data to get convergence and there should be higher accuracy as I reach this point. The accuracy levels off around 700 iterations, which makes me think that convergence might not be possible at all with this activation function.

As more data is added testing data accuracy increases, training accuracy decreases. It levels off quickly around 1,000 samples and stays around .75 accuracy for both the test and training set. This implies that neural network learns from the dataset with a small subset and does not improve with more data.

Support Vector Machines (SVM)

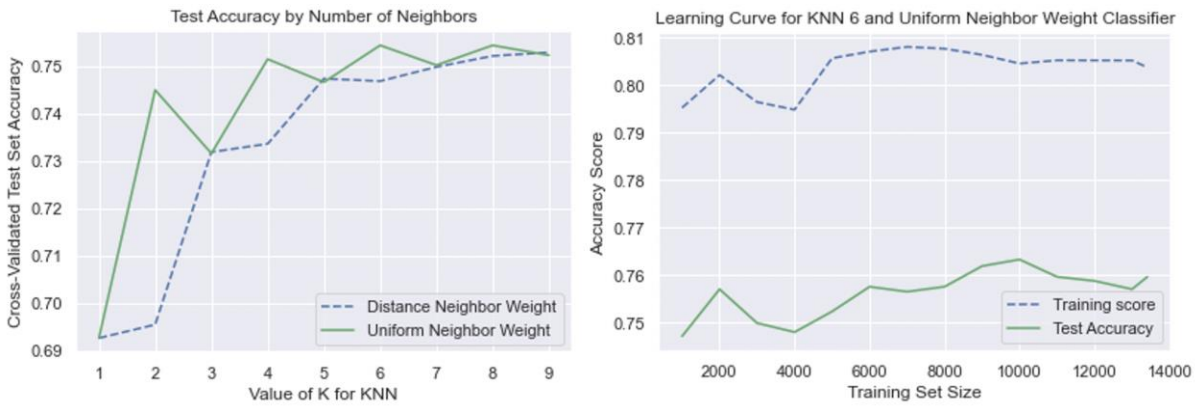


More iterations did not improve the performance of the classifier on the second dataset either. Only one iteration is necessary for kernel convergence.

For the poly kernel svm, as the training set size increases, the testing accuracy increases, and the training accuracy decreases. For the linear kernel as the training set size increases, the testing and training accuracy goes down. The testing dataset's accuracy levels off quickly, only 2,000 training samples. The accuracy on the training set goes down but does increase and decrease slightly as the training set gets large.

Again, we see that the poly kernel has a higher accuracy than the linear kernel. I think that it's because the data is not linearly separable, and the poly kernel fits the data better.

K Nearest Neighbors



I also used cross validation to see what the best neighbor weight measure and number of neighbors were. Calculating the target using six neighbors was also the best number of neighbors for this dataset as well. At nine neighbors, distance weight for neighbors started to perform like the uniform weight. It might be possible that with more neighbors with a distance weight, I could get comparable performance to the uniform weight and 6 neighbors. Since uniform weight is the least complicated of the two weight measures and 6 neighbor the smallest amount with the best performance, I did not think exploring that question was worth it.

Adding more data to the train on did not increase the accuracy of the classifier like I thought it would. 14,000 data points is certainly better than one data point, but it only increased the accuracy of the classifier by one percentage point. I have a few theories. New data points just might not be accurately described by its neighbors. The closest neighbor could obscure the importance of certain features that are more predictive than others so just calculating a uniform distance measure for a vector of all the features may not actually effectively describe data point relationships with each other, so when more data is added the “closer” data point are not good predictors.

Comparative Analysis

	Classifier Name	Training Error	Testing Error	Testing Accuracy	Training Time
0	K Nearest Neighbor	0.196495	0.242171	0.757829	0.014311
0	Support Vector Machine	0.218195	0.219555	0.780445	5.477254
0	Gradient Boosting Tree	0.195078	0.197112	0.802888	0.803089
0	Neural Network	0.249217	0.249652	0.750348	2.189612
0	Decision Tree	0.202908	0.198156	0.801844	0.052785

The kind of accuracy that classifiers were able to achieve on this data set was lower than what the bank marketing dataset was able to achieve. There were fewer training examples, and a higher proportion of the dataset was the positive of the target variable, 25%. K nearest neighbors and neural network would immediately be thrown out because I could get about the same error rate if I just guess 0 for all examples. In addition, neural network took the second longest amount of time to train. SVM I would not choose because it is not the most accurate and it takes the longest to train. Although gradient boosting tree is the most accurate, I would not choose it over decision tree. Again, I think that decision trees are better able to answer the kind of direct questions this classification problem wanted answered Looking at things like feature importance or even constructing the decision tree, I can tell exactly which features are driving the overall performance of the model. I think the kind of analysis you can do with one decision tree and the overall explainability of the algorithm is more important than having a slightly more accurate algorithm that a gradient boosting tree gives you. And it takes less time to train than the gradient boosting tree. You can get some of the same metrics from the gradient boosting tree classifier that you can from the decision tree but if similar performance is reached with something less complicated, I go with that.