Supervised Learning for Credit Card Default and Bank Marketing Campaign

<u>Background:</u>

I strongly believe machine learning will completely revolutionize the banking industry and as a result, bring significant benefits to our economy. For example, as banks become more accurate at predicting defaults and bankruptcies, they can increasingly offer lower interest loans and special offerings to "safer" clients driving down risk, attracting more clients, and gaining a competitive advantage over their competitors. This will drive an analytical arms race between major banking organization to develop more accurate predictive models, decreasing overall interest rates, and increasing the amount of loans and special offers that they approve. These lower interest rates and more abundant loans and special offers will in turn, go directly into the economy, developing new businesses and growing existing ones. As a result, I am fascinated in applying machine learning to banking datasets and would love to enter the banking/financial industry after I graduate from this program. Hence, I picked two data sets related to banking.

First, I am looking at a credit card default dataset that contains 30,000 records of different credit card holders in Taiwan along with whether they defaulted on their payments. For each credit holder, we have information on their credit limit, gender, educational background, marital status, age, and history of past payments and are interested in predicting whether they will default on their payments or not. This dataset is very large with 30,000 data points and 23 columns and can help banks get a better understand on predicting payment defaults.

After developing a model for predicting whether a person will default on a payment, I believe the next biggest focus for banks is attracting potential customers. As a result, for my second dataset, I will be looking at a bank marketing campaign dataset that holds 41188 records of different individuals that were contacted by a Portuguese banking institution and whether they subscribed to a new product offered by this banking institution. For each individual, we have information on their personal information such as age and job as well as information on when and how they were contacted. We want to use this information to predict whether they subscribed to the new service or not. Again, this dataset is very large with 41188 data points and 21 columns and can help banks and other businesses understand how to best communicate with potential customers and clients.

Again, I strongly believe that banking is and will be a major consumer of data science and machine learning and am very interested in applying machine learning to these relevant datasets. Across the next few sections, I will discuss how different models work and talk through how these models functioned on both datasets.

<u>Data Pre-processing:</u>

For both the payment default and the bank marketing data, the first thing we want to do is pre-processing the data to make it ready for analysis. First, we can check if there are any null values in our data. If there are, we can do further analysis on whether we should fill in the null values with averages, medians, regression or dropping the rows completely. Fortunately, both datasets are complete and there are no missing values.

Next, we want to look for any categorical variables in the datasets. One easy way to do this by looking at the unique values in each column. For example, for the payment default dataset, there are 30,000 data points and only 4 unique values for the marriage column. Looking through these unique values shows that they are indeed categorical variables. After finding all the categorical variables, we want to create n – 1 dummy variables for each column where n is the number of unique categories in that column. For the credit card default data, we created categorical variables for the education and marriage columns. For the bank marketing data, we created categorical variables for job, marital, education, default, housing, loan, contact, month, and day_of_week.

Now that we can ensured that we have the proper data and dummy variables, we can randomly split the data in training and test sets. Initially, we will follow the python default split of 75% training and 25% test. After splitting the data, we will now use the training data to develop our models. First, we must investigate data imbalance. In a case like credit card default, the number of defaults will be significantly less than the number of non-defaults. Similarly, for the bank marketing data, the number of people to accept the new deal will be significantly less than the number of people who won't. As a result, it is important that we resample our data to ensure there is an equal or near equal count of data with different classes such as default vs. non-default, accept over vs. reject, etc. Otherwise, we may have a model that only predicts the dominate class without predicting any other classes but still performs well across all data.

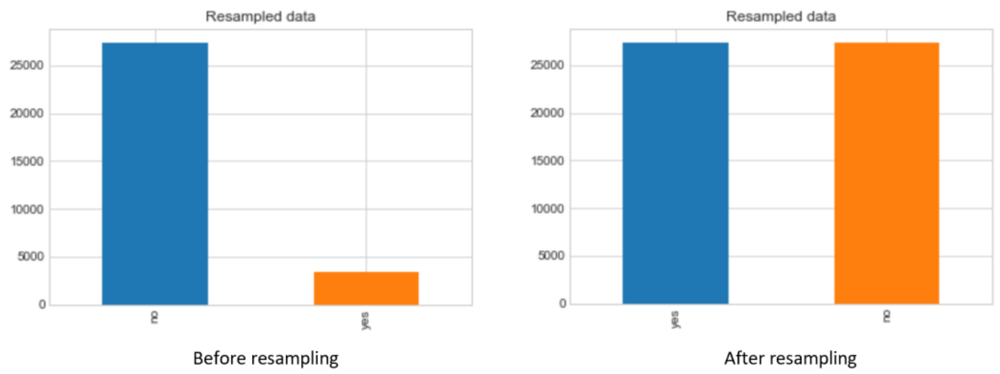


Figure 1. Resampling Data for Default Data

Finally, as a last step for preprocessing, we can scale our training and test datasets. Scaling is especially helpful for KNN, ANN, and SVM and does not hurt boosting or classification. As a result, let's do this first before we start with the different methods. And now that we are finished with preprocessing, we can start with our different models. In the following sections, I will describe each method and how it affects both datasets.

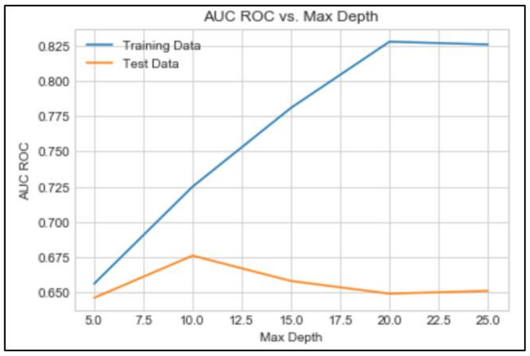
Decision Trees

The first algorithm is the decision tree. This algorithm is a greedy algorithm that creates decision splits based on criteria for certain columns to classify data into different categories. For example, if the person is over 35, we predict that this person will default, otherwise, we need to check his income. This algorithm is great for understanding the logic used to classify data into certain categories. This is great for something like banking where they have to explain exactly how their model made its classification decision and why for example, they can't give you this loan.

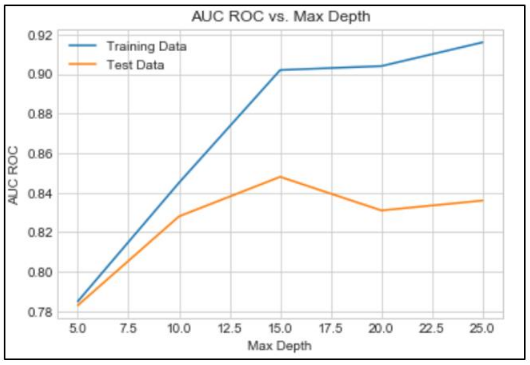
Grid Search:

First, we need to find the best hyperparameters for our decision tree. We can accomplish this by tuning our model with different values for different parameters using a grid search and finding the hyper parameters that give the best results. We will tune our decision tree and our other models with a 10-fold Cross Validation (CV) based on the Receiver Operating Characteristic (ROC) Area Under Curve (AUC). A 10-fold CV will break the training data randomly into 10 equal-sized pieces and use 9 pieces as the training data and 1 piece as the validation data. It will then rotate through the 10 pieces where 9 of the pieces are used as training and 1 is used as the validation. The ROC is curve created by plotting the true positive and false positive rates of a model at different threshold values and the area under the curve determines the strength of the model. We want to maximize the ROC AUC to get the best model.

The hyperparameters we will look at tuning will be the max features, min sample leaf, and max depth. Our first parameter is the max features. This value is important as it tells the model how many parameters to look at for each split. The next parameters, min sample leaf and max depth, are parameters that help with tree pruning and preventing overfitting. The min sample leaf determines the min number of samples in each leaf. A higher min sample leaf value means there needs to be more samples in each leaf and will reduce the number of splits. Similarly, the max depth sets the maximum tree depth for how far the tree can grow. This prunes the tree and prevents overfitting. As an experiment to prove this, we can try use our best parameters and vary the max depth of the model to understand how this affects model performance. We can see the results in figure 2 which show the AUC as a function of max depth for default and banking marketing data respectively.



Default Data



Bank Marketing Data

Figure 2. Max Depth of Tree vs. AUC for Default and Bank Marketing Data

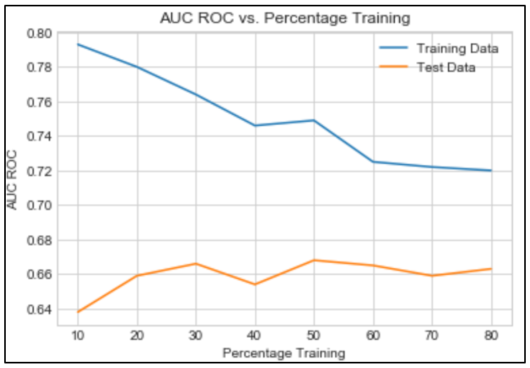
As we can see in figure 2, with larger max depth, our models start fitting the training set better and better. However, at the same time, there is a drop in model performance on our test datasets. This is because the model is overfitting to the training set on random noise that are not present in the test set. As a result, we can prune our decision trees and prevent overfitting by setting our max depth.

After running our grid search, we see that the best parameters for the default dataset are max depth = 10, max features = log2, and min sample leaf = 10. These parameters make a lot of sense as both max depth and min sample leaf are used for pruning and max features doesn’t really matter as there are about 20 features and log2(20) = sqrt(20) = 4 features.

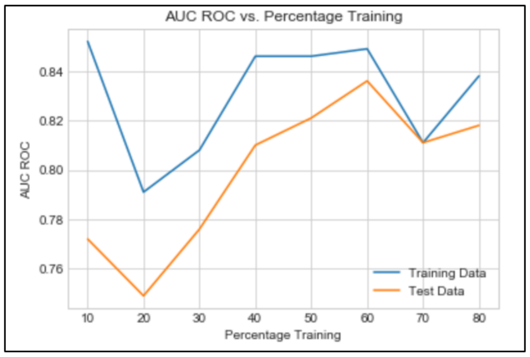
For the bank marketing data, we see that the best parameters are max depth = 10, max feature = sqrt, and min sample leaf = 10.

<u>Results:</u>

Now that we have good models, we can look at their performance based on different percentage splits of test vs. training. We can see in figure 3 that as we get higher percentage of training data, both of our models increase in performance on our test data. This is because increasing the size of the training data decreases overfitting on the training data as noise in the data cancels each other out and the model will focus on modeling the actual trends. As a result, we have higher performances on the test data as we have more training data



Default Data



Bank Marketing Data

Figure 2. AUC ROC vs. Percentage Training

From the data above, we can see, for both cases, that the testing data AUC is very close to the training data AUC. However, there is still a gap between the two lines. This is a sign that there is maybe some more overfitting that we could work on to improve.

Neural Networks

The second algorithm is the Artificial Neural Network (ANN). This algorithm composed of layers of neurons which take weighted inputs and output a result based on an activation method. ANNs are great for determining relevant features in large datasets. However, ANNs are "blackboxes" and it can be very difficult for users to understand how decisions were made.

<u>Grid Search:</u>

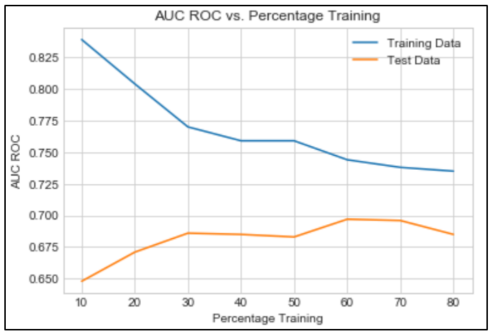
Similarly, we will use a grid search to identify the best parameters. The parameters we will be tuning are the learning rate and hidden layer size. The learning rate is used to update the weights within the neural network. They are essentially the step size to take to ultimately minimize error. If our learning rate is too high, we can overshoot the minimum and if our learning rate is too low, it could take a lot longer to reach the minimum. Next, we want to vary the hidden layer size, this is the number of layers there are in our neural network. Generally, we would want less than 4 layers and we need less nodes with more layers.

From our results, we see the most optimal hyper parameters for the default dataset are learning rate = .01 and hidden layer = (13,7). Again, these parameters make a lot of sense, learning rates are generally somewhere between .001 and .1 and there should generally be less than 4 hidden layers and a good estimate for the number of nodes in each layer should be half of the number of inputs + outputs.

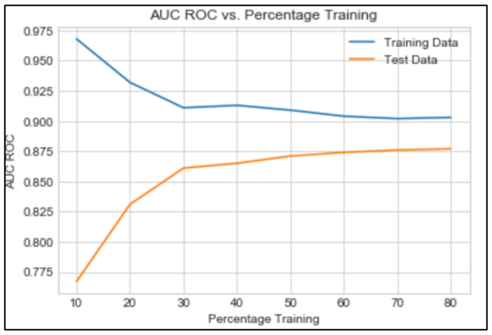
For our bank marketing data, the optimal hyper parameters are learning rate = .01 and hidden layer = (35, 18, 9). Again, these hyper parameters make a lot of sense.

<u>Results:</u>

Now that we have a good model, we can look at its performance on different percentage splits of test vs. training. Like our decision tree, increasing the percentage of training data decreases performance on our training data and increases performance on our test data. Similarly, this is because increasing the size of the training data decreases overfitting on the training data as noise in the data cancels each other out and the model will focus on modeling the actual trends. In the graphs, below, we can see our default model can hit almost .7 AUC at 60%-70% training data and our banking marketing data can hit .875 AUC.



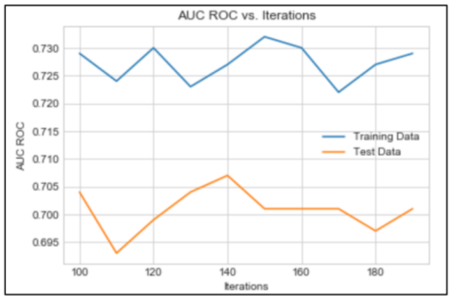
Default Data



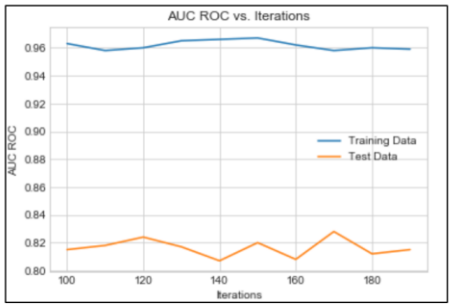
Bank Marketing Data

Figure 3. AUC ROC vs. Percentage Training

As this is also an iterative algorithm, we can also look at the performance vs. the number of iterations. After testing different values for iteration, it seems that for both models, the model does not converge until the max iteration is more than 100. However, as we see in figure 4, the max iterations do not seem to influence the performance of the models once it is high enough for the models to converge. This is because the models will stop after it converges so higher max iterations does not matter for ANN. In both cases, the models have a learning rate of .01 and needs at least 100 max iterations to converge.



Default Data



Bank Marketing Data

Figure 4. AUC ROC vs. Max Iterations

Boosting

The third algorithm is the boosted algorithm. This algorithm is a voting algorithm that takes multiple weighted weak learners and have them vote on the outcome. We are doing this with multiple decision trees and are essentially creating new trees to correct the remaining residual error.

Grid Search:

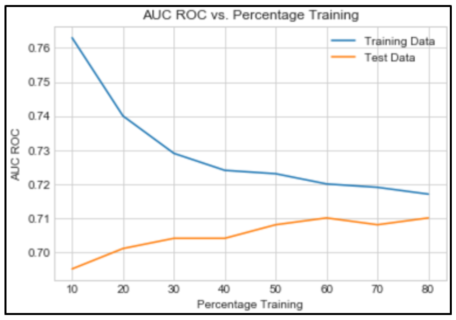
Again, we will use a grid search to identify the best parameters. The parameters we will be tuning are the learning rate and n_estimators. The learning rate is similar to the weight factor and slows down the learning rate of each tree to prevents overfitting. To compensate, given a small learning rate, we will need high number of estimators to ensure proper fit. These two parameters work together to slow down the model's learning speed and prevent overfitting.

Looking at the results for our tuning, we see the most optimal hyper parameters for the default data are learning rate = .01 and n_estimators = 500. Again, these parameters make a lot of sense. Learning rates are generally smaller with a large n_estimators to compensate.

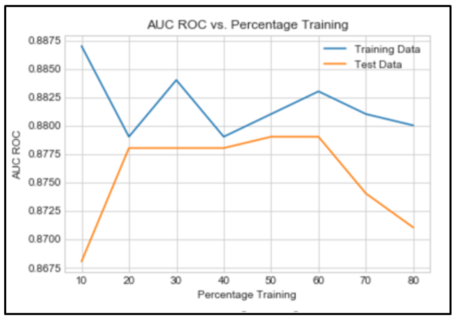
For the bank marketing data, the most optimal parameters are learning rate = .01 and n_estimators = 1000.

Results:

Now that we have some good model parameters, we can look at our model performance on different percentage splits of test vs. training. For the default dataset, increasing the percentage of training data decreases performance on our training data and increases performance on our test data. For the bank marketing data, the AUC stays stationary around .877



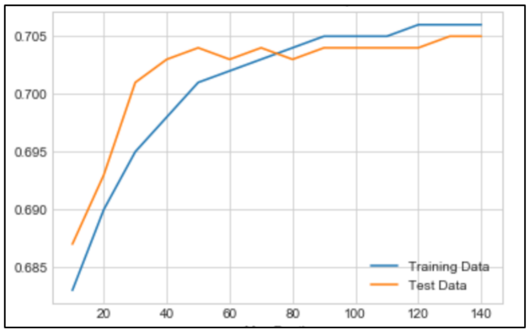
Default Data

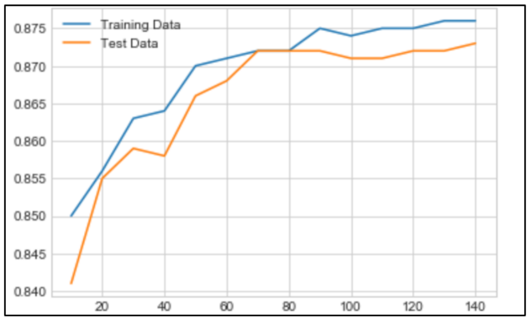


Bank Marketing Data

Figure 5. ROC AUC vs. Percentage Training

Furthermore, boosting is an iterative algorithm where we build out additional decision tree to compensate for error. As a result, we can also look at the performance vs. the n_estimator. As we see in figure 6 for both cases, the more estimators we use, the better the model performs on both the training and test set. This is different from the ANN where increasing the number of iterations does not do anything once the model converges. This is because for boosting, as we iterate through more estimators, we cover more of the edge cases, allowing the model to predict better and better.



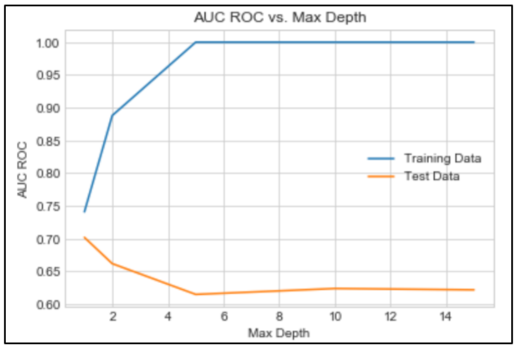


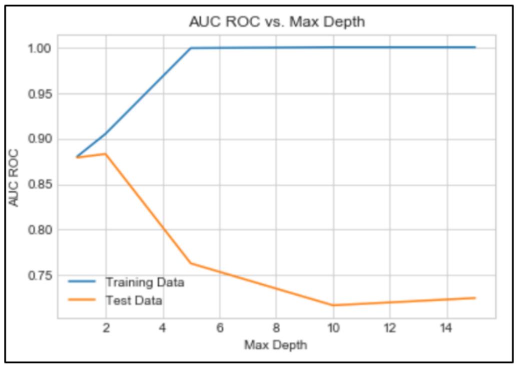
Default Data

Bank Marketing Data

Figure 6. ROC AUC vs. Number of Estimators

Finally, we will use using the max depth of our decision trees for pruning. By default, each decision tree in our boosted model has a max depth of 1. This max depth sets how far the trees can grow. This prunes the trees and prevents overfitting. As an experiment to prove this, we can try use our best parameters and vary the max depth of the decisions trees to understand how this affects model performance.





Default Data

Bank Marketing Data

Figure7. ROC AUC vs. Max Depth

As we can see in figure 1, for both models, as the max depth increase, the AUC for the test data immediately approaches 1. At the same time, the AUC for the training data decreases. This is a sign of significant overfitting. As a result, we can prune our decision tree and prevent overfitting by setting the decision tree max depth.

Support Vector Machines

The fourth algorithm is the Support Vector Machines (SVM). This algorithm plots all data points in an n dimensional space and develops an optimal separator in this space that best classifies different types of classes and maximizes the minimum margin from all data to the separator. Because the minimum margin is dependent on all the columns, it is important to normalize the data before using this model to ensure one column does not dominate all of the other columns.

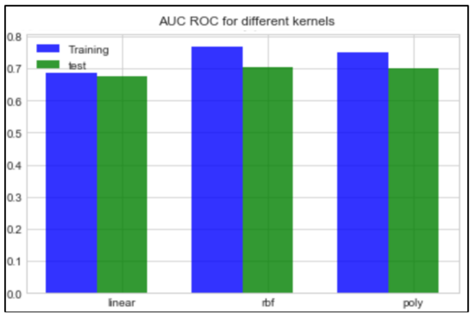
Grid Search:

First, we will use a grid search to identify the best parameters. The parameters we will be tuning are the C value and the types of kernels. The C value balances the training error and the test error. It dictates whether we are looking for a separator that creates the maximum minimum margin or one that best classifies all the training data points correctly. The kernel can be one of a few options given by the model. For this problem, we will try three kernel functions to show that we can easily swap out the kernels.

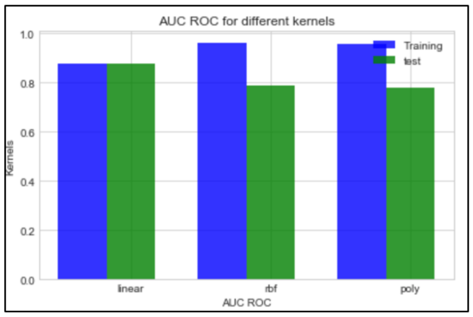
From our results, we see the most optimal hyper parameter for default is at c = 8. This large c values shows that this model performs better when it is developed to focus on properly classifying the different classes.

Similarly, the most optimal c value for the bank marketing data is also at c = 8. This shows that both models favor correctly classifying most of the data rather than maximizing the minimum margin to the separator line.

Below, we can see the performance of models with different kernel types for the default and bank marketing data.



Default Data

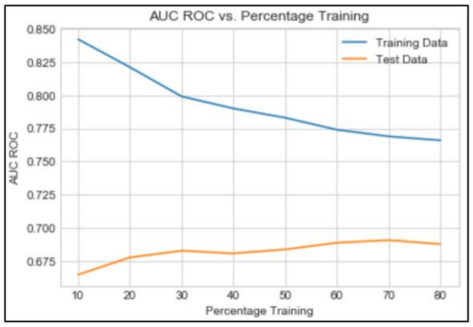


Bank Marketing Data

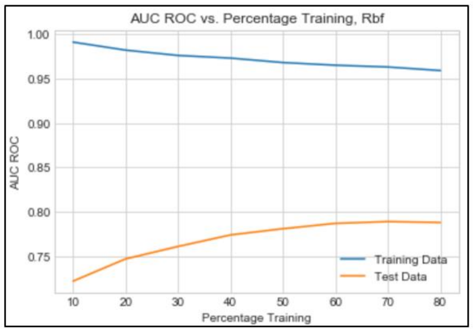
Figure 8. Performance on Test and Training based on Kernel Types

Results:

From here, we can look at our results. Just like all the other models we have looked at so far, the more training data we have, the better our models perform on the test data. Figures 9 and 10 shows the model performance for different training % for the RBF kernel and the Poly kernel respectively. In the graphs below, we can see our default model can hit almost .7 AUC at 60%-70% training data for the poly kernel and our banking marketing data can hit almost .8 AUC at 70% training data.

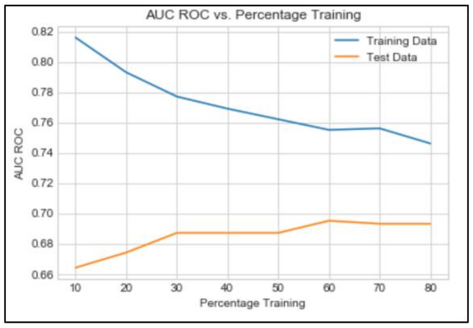


Default Data

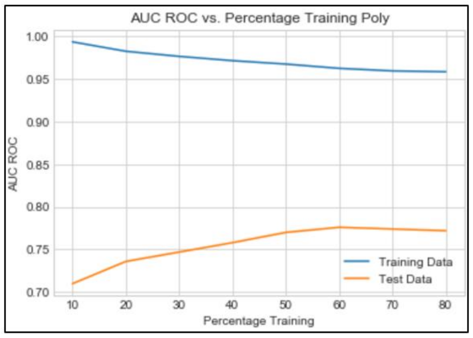


Bank Marketing Data

Figure 9. ROC AUC vs. Percentage Training for RBF Kernel



Default Data



Bank Marketing Data

Figure 10. ROC AUC vs. Percentage Training for Poly Kernel

KNN

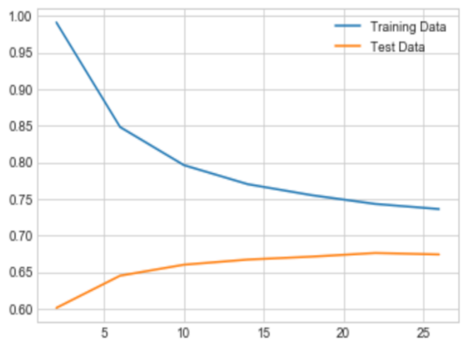
The last algorithm we are looking at is the K Nearest Neighbor algorithm. This algorithm plots all data points in an n dimensional space, where n is the number of columns, and classifies new data points as the dominate classification of the k points closest to the new data points. For example, if k = 5 and 4 out of 5 of the closest colors to a new data point is blue, then the new data point's color will be classified as blue. Because the minimum distance is dependent on all the columns, it is important to normalize the data before using this model to ensure one column does not dominate all the other columns

Grid Search:

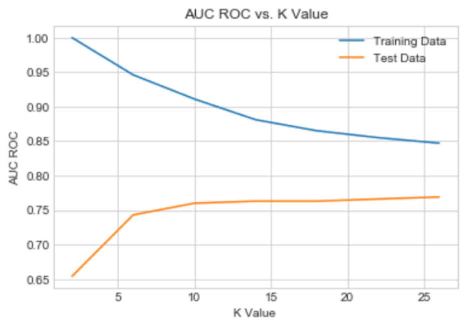
The major parameter for KNN is the number of closest neighbors to consider. Lower numbers of k will create overfitting on the training data, and it is important to find the optimal k value for each model. In this case, we cycled through multiple k values to find which k value will give the best AUC for the test data.

Results:

Figure 10 shows the AUC vs. K Values for the Default and Bank Marketing datasets. When k = 1, our models have a AUC of 1 for both datasets. This means our models correctly predicts every training data point. This happens because the model is the training data and when fitting the training data, the closest 1 neighbor is itself leading to a 100% correct prediction. This is an extreme exampe of overfitting. Furthermore, looking at the graphs below, the best k values for the default and bank marketing data are k =22 an k = 25 respectively.



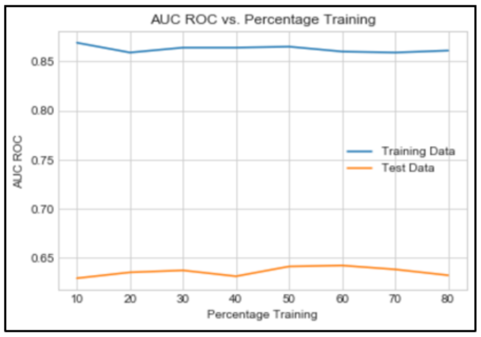
Default Data



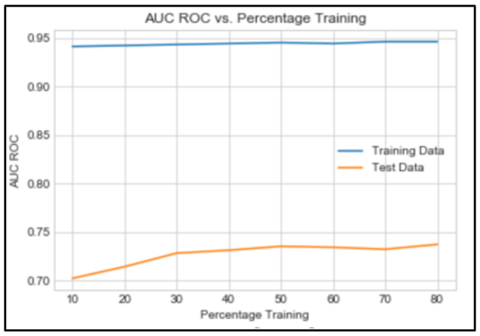
Bank Marketing Data

Figure 10. ROC AUC vs. K Values

From here, we can graph the model performances as a function of percentage training. Unlike the other models, however, the percentage training does not seem to affect the AUC. Of course, this makes a lot of sense for KNN. This is because there is so many data points that even 10% of the such large data sets is enough to create a n-dimensional space where new points can be classifies. In the graphs below, we see the KNN model yields a .63 AUC on the default data and a .73 AUC on the bank marketing data.



Default Data



Bank Marketing Data

Figure 11. ROC AUC vs. Percentage Training

Conclusion:

We had two classification datasets with one focusing on credit card default data and the other focusing on bank marketing data. After conducting 5 different classification algorithms on each dataset, we see that we can achieve the highest ROC AUC using Boosting and ANN. I believe ROC AUC is a great measurement for model performance as it shows how well the model is able to predict true positives and true negatives. We were able to achieve a .71 AUC for the credit card default data with a boosting algorithm and a .88 AUC for the bank marketing data. It makes a lot of sense that the iterative algorithms performed the best. These methods go through many iterations or develop new trees to increase their fidelity.

Out of all the algorithms, it seemed like decision trees were the fastest to train followed by neural networks and boosting. SVM and KNN took the longest to train, especially for the bank marketing dataset. This is because these datasets had many predictors and algorithms such as KNN and SVM struggle with high dimensionality. In fact, including dummy data, the bank marketing dataset has over 50 predictors. This caused SVM and KNN to take over an hour each to train. To combat this, I implemented a PCA on the data for the bank marketing KNN model. I kept 80% of the variance and decreased the number of dimensionalities from 53 to 25. This significantly improved the training speed decreasing the training time from 1 hour+ to about 10 mins. If I had time to make additional changes, I would implement a PCA for all the SVM and KNN models. This would decrease dimensionality and significantly improve these models' improvements. Similarly, I could also go through the data and perform feature selection to remove some of the unnecessary features. This would also increase model performances. Another thing that we can do is further hyper parameter tuning. There is still a difference in model performance on the training set and the test set. This could be caused by overfitting which can be improved with further analysis of the hyper parameters. Finally, we can see that our data sets definitely influenced the model performances. For example, the max AUC for the default data is .71 while the max AUC for the banking marketing data is .88. This shows the bank marketing data is better at developing a model for predicting whether a consumer will accept a new deal.

Overall, we were able to develop multiple models to predict whether a customer will default on credit card payment and whether a person will accept or decline a new promotional deal.