Dylan Floyd

CS 7641

Fall 2018

Assignment 1

Datasets:

Poker Hand Dataset:

This dataset is used to make a multi-variate classification of the type of poker hand generated, given information about the cards used in a Texas Hold 'Em style poker game. There are 25010 data points, each with 10 features that describe the exact amount of information required to classify a poker hand with 100% certainty assuming the rules of the game are already known. There are 5 cards in a poker hand, and each card a single data point is represented by two features that describe suit and rank. Suit is a categorical variable that is represented with integers between [1 – 4], and rank is also represented with integers between [1 – 13]. The label is represented by integers [0 – 9] and designates which of the 10 possible hands those 5 cards make when seen together. The dataset was obtained from the UCI Machine Learning Repository.

Credit Card Default Dataset:

This dataset is used to make a binary classification of whether a default occurs or not, given information about a person's credit card history. There are 30000 data points, each with 23 features such as balance limit, age, sex, marriage, and other information about payment history. All of the 23 features are represented numerically and point to the label for 'default payment next month' which is 1 for yes, and 0 for no. This data was obtained from the UCI Machine Learning Repository.

Interesting Dataset Comparisons:

In poker, the number 2,598,960 represents the number of unique 5 card hand combinations, where the order of the cards does not matter. Meaning the number of possible data points is finite with the poker dataset, which doesn't exactly seem like as large of a number compared to the theoretically limitless number of data points possible with the credit card default dataset.

The poker dataset is not practical, but the credit card default dataset is relevant for the work that banks and credit card companies require to stay in business. It is unrealistic to think a function with simple if-statements and other conditionals can be created and predict credit card default in real life situations with 100% accuracy; not only is that realistic for poker, it's probably used in practice with online games. However, this doesn't mean that the analysis on the poker dataset isn't interesting or important. The poker dataset's performance under a variety of learning conditions can help illustrate whether or not these algorithms are capable of producing a model that infers the rules of a game which it knows nothing about.

Describing what a poker hand is given 5 cards is an easy problem for a human, but difficult for a computer. On the other hand, predicting credit card payment defaults with high accuracy is probably much more challenging for a human it is for a computer. Particularly with these algorithms, one aspect that will probably make the poker dataset difficult is that the order of the cards doesn't matter. Consider two royal flushes of the exact same suit. For the first royal flush hand, card C1 in the dataset could represent the 'Ace' but in the second flush hand the 'Ace' could be represented by card C4.

Another interesting comparison is the disparity in training examples for each class. In the poker dataset, there are fewer examples of the rarest types of hands which reflects the nature of the game itself. Rare hands have more value because only a much smaller subset of cards can be combined to create them. Taking a royal flush for example again, it is the most valuable hand in the game and it requires having the top 5 cards by rank and all cards being the same suit. The rare hands should be expected to be much harder to predict than the common hands like nothing but a high card or a single pair. With the default dataset, the least represented of the two classes makes up roughly 22% (6636 of 30000) of the total data points. Only two of the most commonly represented classes (high card and a single pair) make up more than 22% of the total data points.

Methodology:

Five different supervised learning algorithms will be used to create models that make predictions for classifying data points in both the poker and default datasets. Two different data sets are used so that accuracy can be compared not only across algorithms but also across models. Training accuracy, cross-validation accuracy, testing accuracy, as well as performance times are recorded and analyzed. There might be one or several hyper parameters that impact model performance, and initially a list of values will be set for each hyper parameter. Multiple learners will be instantiated to loop over those range of values for a particular hyper parameter, while holding all other hyper parameters to their default values. Model complexity plots are created to visually examine the impact on accuracy for multiple values of a hyper parameter. Some values will cause the models to generalize better to new data, others will have the opposite effect, and some will seem to have little effect at all. This process is repeated 5 times to average out the results. Using more than 5 iterations would be appropriate, but is quite time consuming. The ideal hyper parameter value is one that produces the best balance between underfitting and overfitting the training data, and it will be passed into a new learner that's instantiated with a conglomerate of the other individually optimized hyper parameters. Next, this final set of hyper parameters are used to create learning curve plots by looping over a list of training set sizes as percentages.

At this stage, information from the model complexity plots can be used in conjunction with the learning curve plots to make educated adjustments to the hyper parameters. When used together, the hyper parameters may improve the model, make the predictions less accurate, or make no difference at all if say, one hyper parameter is dominating the impact. It's worth taking a couple of extra looks to examine how they perform together in case some minor adjustments are capable of helping the model generalize better to never before seen data. To avoid bias towards test score results, the gauges for making adjustments came from the training accuracy and the cross-validation accuracy scores. By resampling from training data to make validation sets and not actually using new data, the cross-validation accuracy provides better insight into what probably matters most with these types of problems: how effectively will a model actually generalize to new data. The learning curves helps illustrate that the number of available training data points has on accuracy . The amount of time it takes to train and query a model should always be considered as well, especially in scenarios where model complexity is increasing but the gains in how well the model generalizes to new data are limited.

Decision Trees:

The decision tree will partition training data from a parent node into two separate nodes based on the feature that provides the greatest information gain at that node. This process is repeated many times and the tree can become quite large and complex. Figures 1-3 demonstrate that pruning can be an effective way to reduce a decision tree's tendency to overfit the training data.

Poker Dataset:

Figure 1.

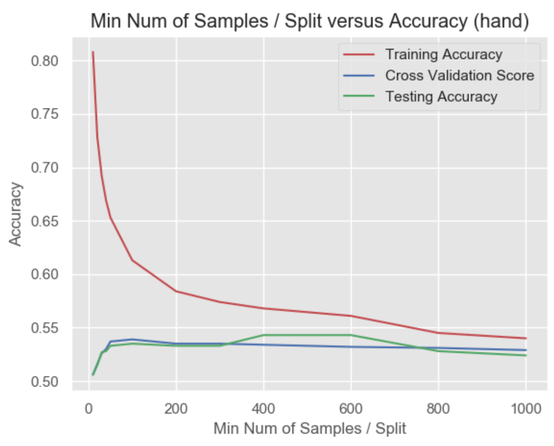
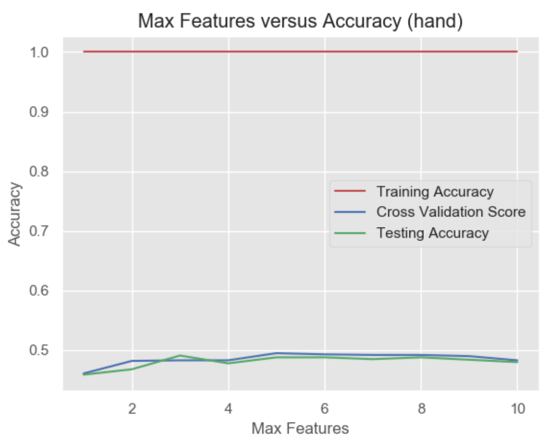


Figure 2.



The parameter, min_samples_split, enforces the number of data points that must be present in a node in order to perform a split. This particular parameter has no effect on the size of the leaf nodes that are split from the parent node, but it is a form of pruning that highlights a decision tree's tendency to overfit with small values as seen in Figure 1. Too large of a value, however, can force the tree to underfit the model, which is also seen in Figure 1. Performing poorly on both training and cross validation sets points to underfitting. The training accuracy drops exponentially from over 0.80 to under 0.55. Even though the distance between training and cross validation accuracy is the smallest at very large values for this parameter, the accuracy values themselves are not only bad but they are decreasing.

The cross validation accuracy is still increasing up until the minimum data points needed to split a node exceeds a value of 100. At that point, the model switches from overfitting to underfitting, so 100 was selected as the value for the hyper parameter in Figure 1 in order to meet in the middle and hopefully reduce error.

The parameter, max_features, plotted in Figure 2 demonstrates consistently high overfitting for all values of the variable that were tested. No optimal value was chosen for this because there are no strong indicators that suggest taking a max_features pruning approach will help the decision tree generalize to new data more accurately. A discrepancy like this means that the tree is probably very large and complex, and also good at predicting labels for data points that it has already seen, but very inaccurate on new data points.

Figure 3.

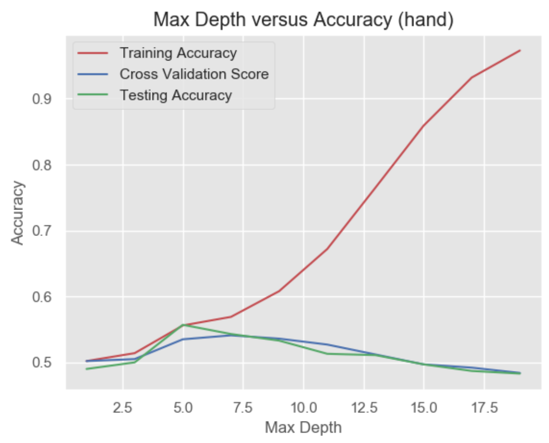


Figure 4.

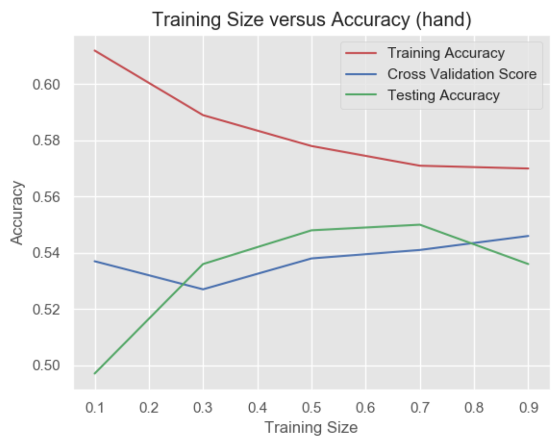


Figure 3 illustrates the pruning effect of max_depth on a decision tree. The tree becomes more complex as the max_depth is allowed to be a larger value and the discrepancy between training and cross validation accuracy grows rapidly. The ideal parameter here is 7 although the accuracy itself having an average around 0.55 is still pretty low.

These results are not surprising because a decision tree is not well equipped to handle the types of relationships that need to be inferred. Because the decision tree is splitting on one attribute at a time, it cannot easily be trained on numerical data about card rank when the order of the cards doesn't matter.

Credit Card Default Dataset:

In Figure 5, the min_samples_split hyper parameter once again demonstrates an un-pruned decision tree's tendency to overfit the training data. Having small values for the required number of data points to perform a split enables the tree to keep splitting beyond the point where valuable information gains exist, and it starts treating noisy features as important signals. The short distance between the training and cross validation accuracy suggests that the model generalizes very well to new data, which the testing accuracy line confirms. There's probably a safe range of values to choose for min_samples_split between 500 and 900 that minimize overfitting. In this case, 600 was chosen.

Figure 6 is very reminiscent of Figure 2 from the poker dataset, in the sense that very little information is gained by adjust this feature over this range of values. No ideal value was chosen for max_features as a result.

Figure 5.

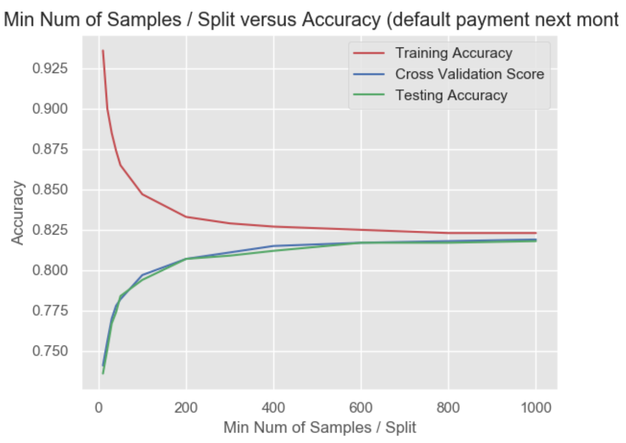
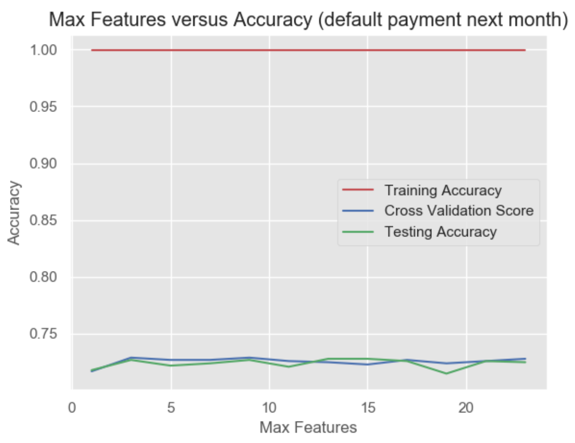


Figure 6.



Max_depth from Figure 7 is also very similar to the trends seen in max_depth in Figure 3 for the poker dataset. The ideal value chosen here was 5 because at larger values of max_depth the model begins overfitting the training data. This is seen by the rapid rise in training accuracy and steady decline of cross validation accuracy. Figure 8 seems to illustrate that a majority of the accuracy is achieved using just 30% of the training examples, but the slope seems to taper off quickly and the model only achieves gradual improvement as training size increases. Overall it seems the model generalizes well for most training size percentages.

Figure 7.

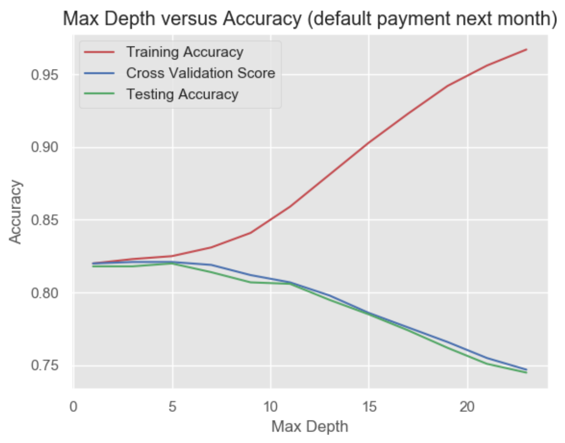
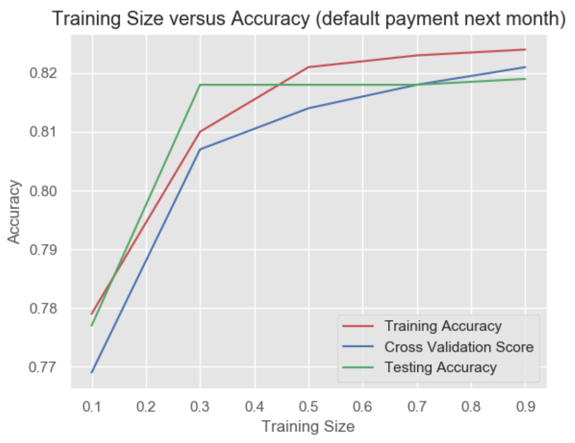


Figure 8.



Boosting:

Boosting is a way to create an ensemble of classifiers that when aggregated, can make a more accurate model than the individual classifiers on their own. The classifier that was fed into this model was a decision tree learner, with the same parameters used in the previous decision tree section since they performed the best on their respective datasets. The AdaBoost Classifier from sklearn was used to see how this type of algorithm responds to different numbers of estimators in the learner.

Poker Dataset:

In Figure 9, the poker dataset continues to confuse the boosted decision tree learner. Almost all of the cross validation and testing accuracy falls below the 0.50 line, but a single decision tree learner with the same parameters lies above 0.50 for almost all values of both of the same curves. The cross validation accuracy seems to be a good estimate for where the test accuracy will be in this case, but the training accuracy itself is much higher than both at several points. Despite the large discrepancy between training and cross validation accuracy, this algorithm with these hyper parameters points to underfitting because of the poor performance overall. Selecting 15 as the ideal hyper parameter could have been a reasonable choice to minimize the distance between training and cross validation while accuracy is trending upwards, and 100 could have been a reasonable choice to maximize accuracy scores. The value selected for the hyper parameter n_estimators used in the learning curve was 60 because it fell just in the middle of those two options. It's not clear how what difference another value in that range would have made with such poor model performance.

Figure 10 shows that the discrepancy between training accuracy and cross validation accuracy decreases as the training set size % increases. Very similar trend lines for this AdaBoost classifier were seen on the learning curve for a single decision tree with the same parameters in Figure 4. These models also demonstrate increasing rates on runtime as n_estimators increases. It's possible that more aggressive pruning should have been used, but the likely culprit for the poor performance is the use of many decision trees estimators where a different algorithm could be more effective.

Figure 9.

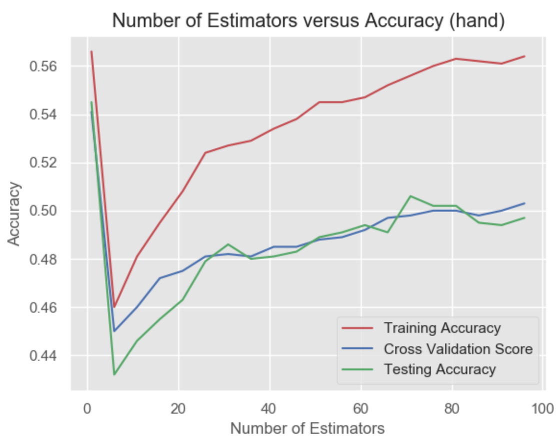
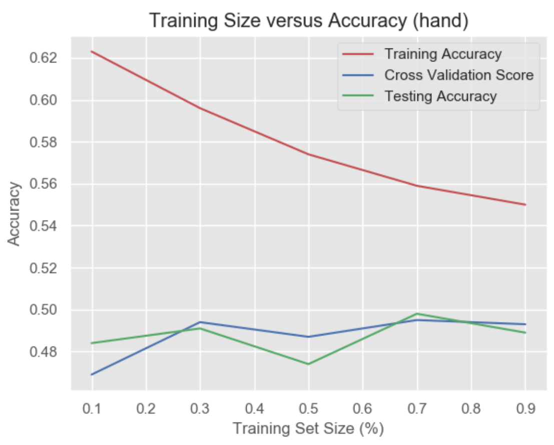


Figure 10.



n_estimators vs Time (s):

1	6	11	16	21	31	36	41	46	51	56	61	66	71	76	81	86	91	96	61
3.3	14	25	43	48	64	75	90	110	104	121	141	151	164	167	179	184	202	221	226

Training set size % vs Time:

0.1	0.3	0.5	0.7	0.9
21.308	51.092	67.259	94.936	120.993

Credit Card Default Dataset:

The training accuracy and the cross validation score are trending in opposite directions as n_estimators increases. Because the values kept diverging, larger values of n_estimators were chosen here to see if the pattern reverses at any point. The results show that the training and cross validation curves never converge as n_estimators is growing. The cross validation error proves to be a good measure for predicting the testing accuracy over mode, but the amount of overfitting becomes more dramatic as n_estimators increases. The optimal value chosen for n_estimators in this case is 2, and that value was passed into the AdaBoost Classifiers at various training set sizes to create the learning curve in Figure 12. It's important to notice how much less overfitting is going on in Figure 12 compared to Figure 11. All of the cross validation scores in Figure 11 are below 0.80 after n_estimators exceeds 100, but nearly every cross validation score is above 0.80 in Figure 12 and the gap between it and the training accuracy curve rarely exceeds half of 1%. Since n_estimators is 2, the training set size will have less of an impact on runtimes than it will for the same algorithm on the poker dataset which has a much larger n_estimators value.

Figure 11.

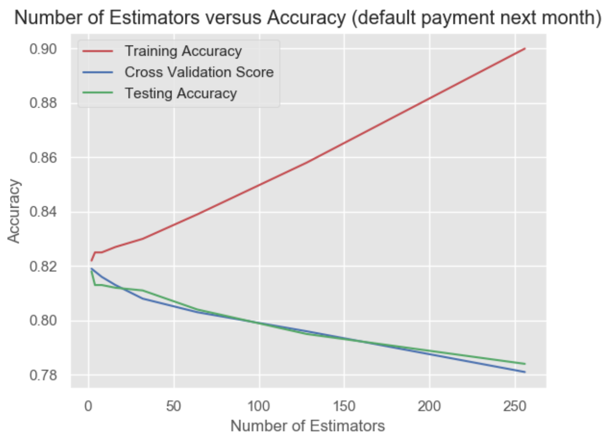
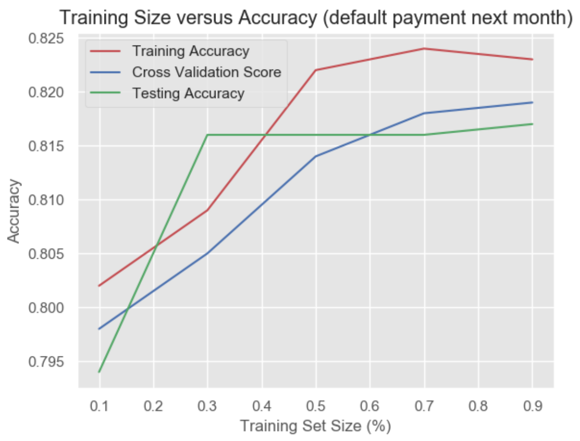


Figure 12.



n_estimators vs Time (s):

2	4	8	16	32	64	128	256
17.74	38.57	68.63	133.0	277.8	564.4	1088.4	2154.63

Training set size % vs Time (s):

0.1	0.3	0.5	0.7	0.9
1.64	5.47	9.49	11.53	14.55

Neural Networks:

In order to cut down on complexity and runtime, the number of iterations was reduced to one so that exploring the parameters could be done faster. In terms of comparing the accuracy and runtime metrics across algorithms and datasets, the only difference that might matter is that the variation might not be captured as well given that the experiment is run once instead of 5 times and averaged.

Poker Dataset:

The benefit of adding additional layers in the neural network classifier seems to plateau after the number of hidden layers exceeds 4. At this point, both the training accuracy and the cross validation accuracy hit respective peaks. The neural networks that make up the model complexity plot produced higher accuracy rates across the board than any of the other supervised learning algorithms did on the poker dataset. The reason for this is that neural networks are more capable of capturing complex relationships between features. were able to demonstrate on the poker dataset. The number of hidden layers drastically increased the runtime, but the number of units in each layer was kept at 100.

Figure 13.

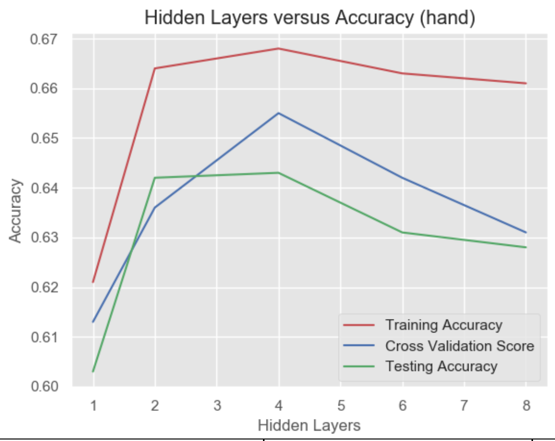
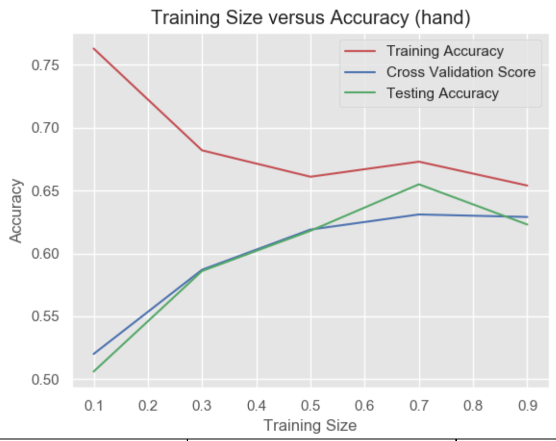


Figure 14.



0.1	0.3	0.5	0.7	0.9
71.35	208.25	327.64	442.62	550.35

Credit Card Default Dataset:
Due to more runtime constraints with the default parameters, the number of units in each hidden layer was decreased to 10 for all models used to create Figure 15. There's a very dramatic positive jump in accuracy between 4 and 5 hidden layers, as seen by the leap from just under 0.25 to just under 0.80. At 8 hidden layers, there is no evidence for increase in accuracy from the experiment, but naturally with more layers the model complexity increased and so did the runtime required to fit and query the neural network. For those reasons, the value selected for this hyper parameter was 6. This learning curve was certainly odd because the zig zag patterns but all of the accuracy values are within 1% of each other. Even on the model complexity plot, all three lines were very close, so in that sense this model has been able to generalize the best.

Figure 15.

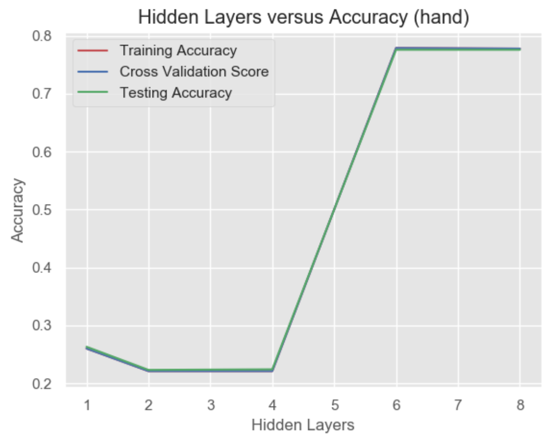
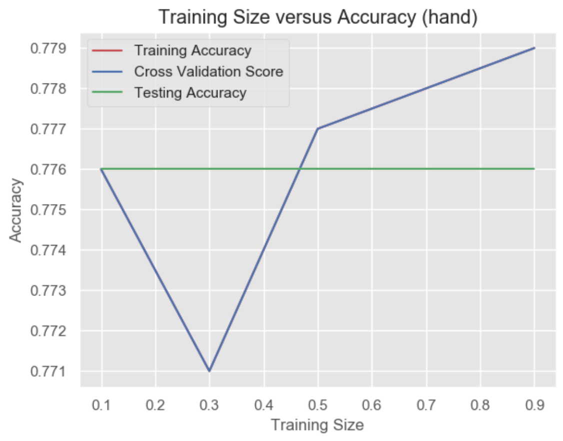


Figure 16.



Support Vector Machines:

Poker Dataset:

This model complexity plot for a SVM with a kernel set to 'rbf' shows that training accuracy and cross validation accuracy increase as gamma increases. If those two trends continue, the training accuracy line has a greater slope than the cross validation error which would normally point to overfitting. In this case, it's actually under fitting because the accuracy scores are so low, and if gamma increases, the gap between training accuracy and testing accuracy will likely grow larger as well. Also as gamma increases, the model complexity and training time the training times increase much faster than other algorithms. The parameter value used to generate the learning curve plot was 0.01, which was selected to minimize model complexity and to sacrifice some of the accuracy in exchange for faster training and querying processes. Seeing a jagged curve for cross validation accuracy but a horizontal line for testing accuracy is odd. I expected the accuracy to gradually increase with more training examples like most of the other plots.

Figure 17.

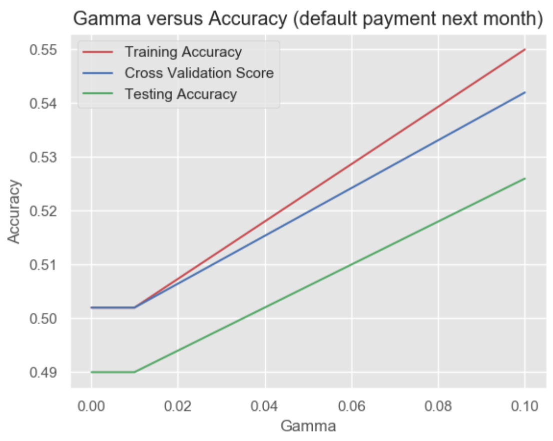
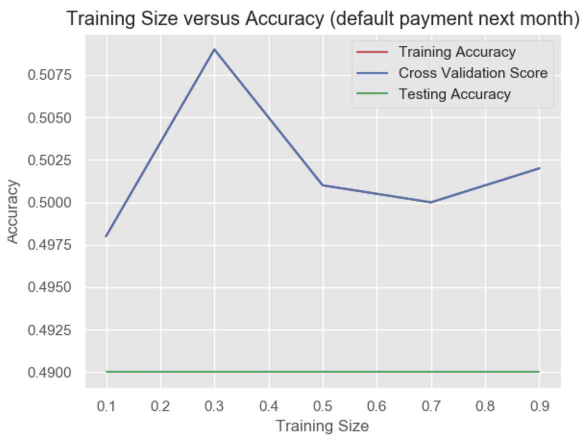


Figure 18.



gamma vs Time (s):

0.0001	0.001	0.01	0.1
806.8	823.0	897.0	1396.1

Training size vs time, with gamma= 0.01, kernel='rbf':
[5.855, 44.886, 121.676, 231.212, 376.471]

0.1	0.3	0.5	0.7	0.9
21.308	51.092	67.259	94.936	120.993

Credit Card Default Dataset:

This model complexity plot shows the training and cross validation accuracies growing at the same rate as gamma increases. The scale of the x axis makes the growth seem linear, but there was very little growth on the gamma values 0.0001, 0.001, and 0.01, then about a 3% jump in both training and cross validation accuracy when gamma is set to 0.1. If larger values of gamma yielded accuracy scores in the 0.9 + range, then taking the SVM approach with a high gamma should be considered. Since these accuracy values are in line with what other algorithms were able to produce, at much faster rates, I would not favor this model over any others. The biggest drawback is the runtime, which just like in the poker dataset was already longer than most and still grew quickly as gamma increased. It's interesting that the testing accuracy was higher than both training and cross validation accuracy. Despite the longer runtimes, the model does actually generalize to the test set well. A gamma value of 0.1 was selected as the hyper parameter for the learning curve to maximize training and cross validation accuracy. The training curve. Given the learning curve plot, this model needs to be trained on somewhere between 50-70% of the training data. It's important to recall that this training set percentage is being taken out of an already reduced subset of the original training data. Trying to train on 50-70% of over 20,000 data points will drastically increase the runtimes.

Figure 19.

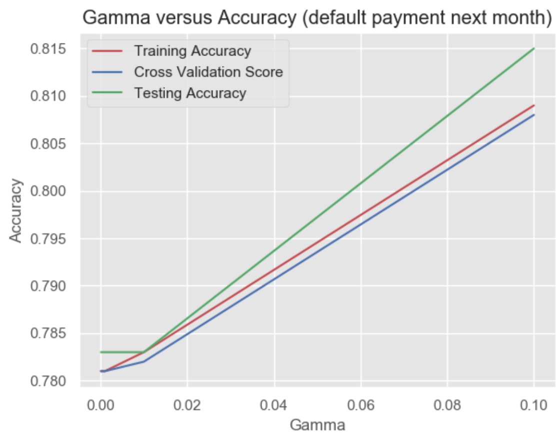
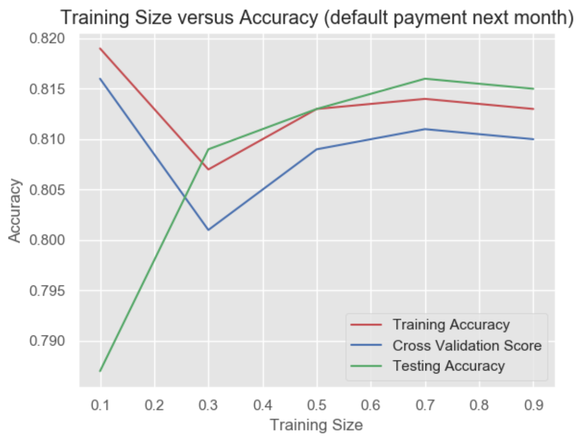


Figure 20.



gamma vs Time (s):

0.0001	0.001	0.01	0.1
79.1	79.6	91.2	102.6

Training set size % vs Time (s):

0.1	0.3	0.5	0.7	0.9
0.92	6.93	23.0	41.7	74.0

k-Nearest Neighbors:

These KNN algorithms will uses a similarity metric between data points in order to make both binary and multi-variate classifications. The default metric is Euclidian distance and the main hyper parameter that was tested is k, which represents the number of closest neighbors to a new data point that will be involved in making the prediction.

Poker Dataset:

Choosing a larger value for k is a way of enforcing that the algorithm attempts to generalize more than it would with a smaller k value. Figure 21 shows that with a really small value for k, the model will perform really well on what it has seen already and worse on unseen data in the cross validation and test sets. The cross validation accuracy scores seem higher than the average for this dataset, and it might be that the algorithm has a better chance to identify a flush (when all suits are the same number) than most other algorithms do because it would help minimize the distance. This algorithm probably struggles with the order of the cards as well as the distance metric between features of two hands. A distance value isn't needed to say that a 9 is a different number than a 4.

Figure 21.

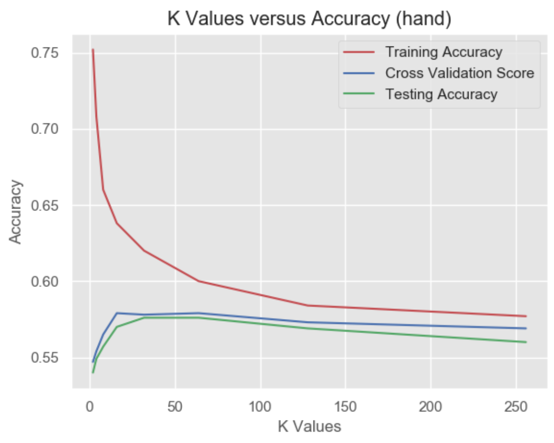
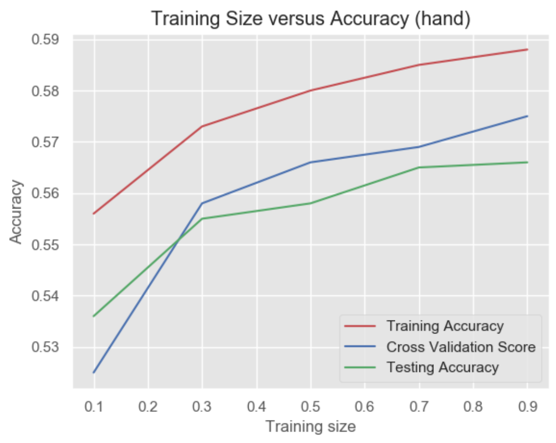


Figure 22.



Training set size % vs Time (s): [2.687, 7.265, 13.969, 18.757, 25.738]

0.1	0.3	0.5	0.7	0.9
2.69	7.26	14.0	18.7	25.7

Credit Card Default Dataset:
Figure 23.

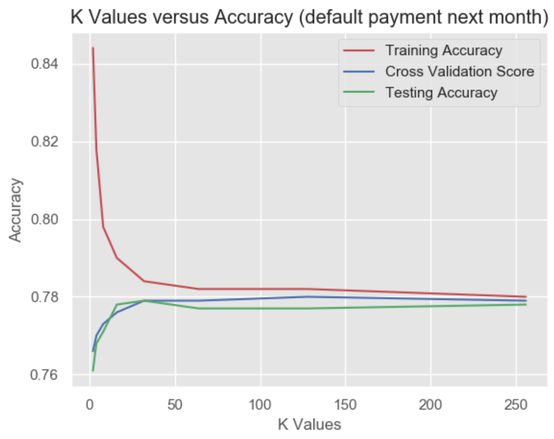
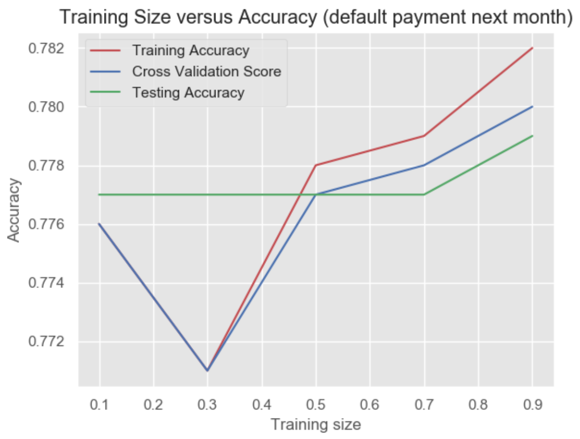


Figure 24.



The model complexity plot in Figure 23 shows a slightly less accurate than average model for making predictions on this dataset. However; there is little space between the training accuracy and cross validation score. The k value that's most likely to produce a model whose generalizes similarly to new test data is somewhere just past 250. However, unlike the poker dataset, this dataset in practice would be updated frequently with new information. The runtimes for training and testing these models is longer with higher values of k because the model complexity is higher as more neighbors are being taken into account for a prediction. The value selected for this hyper parameter was 128 because of a faster runtime. On the learning curve, all of the values for training and cross validation accuracy are basically within 1% of each other. A dip is seen on the smaller training sizes in Figure 24 probably because not enough influential default data points made it into the training set.

Training set size % vs Time (s):

0.1	0.3	0.5	0.7	0.9
4.54	12.31	24.31	43.57	64.67

Additional Reflection:

These datasets are very different from one another, and running a lot of experiments with different hyper parameters for different algorithms proved that. Analyzing the performance of algorithms on two very different datasets was an enjoyable way to learn about how models perform under different conditions. There were some models that performed worse on the credit card default dataset, but most seemed to rest in the low 80's range in their accuracy metrics. The poker dataset would have wound up being less interesting had there not been much better results with the neural network algorithm compared to the other learners. Working with the credit card default dataset seemed to be about discovering how to tweak parameters and make a good model better; working on the poker dataset seemed to be about figuring out how to make a good model out of relationships that most models find very challenging to infer.

Working on this assignment has been eye opening for how nuanced creating a model can be. Examining the relationship between training accuracy and the cross validation score, and factoring in model complexity, learning curves, and runtime metrics is a useful skill for understanding whether or not a model is over or under fitting. Ideally some model exists that is both accurate and precise, and will generalize well to new test data. In practice, editing the hyper parameters seemed like trying to find a balance between those concepts. Consider the scenario where one hyper parameter maximizes the potential accuracy, but has been consistently imprecise. There might be a second option that has a lower expected accuracy but a better chance at maintaining a high level of precision. Finding that balance is a challenging but worthwhile problem.