Supervised Learning Algorithm Analysis

1. Datasets and Methodology

Dataset 1: Poker hands - 25010 instances, 10 features (discrete), 10 classes (unbalanced)

Each instance consists of 10 features corresponding to the suit and rank of each card in a five-card hand. There are 10 classes corresponding to each possible poker hand.

This dataset is interesting in relation to machine learning for two reasons:

- It is a very unbalanced dataset in terms of classes. Royal flushes account for 0.02% of instances whereas "no hand" account for 49.95%.
- There can be multiple interpretations for certain hands. For instance, a royal flush can also be wrongly classified as a straight flush, straight, or a flush.

Intuitively, for these two reasons a classifier is going to have a difficult time correctly classifying certain hands. It will be interesting to see which algorithms handle this best.

Dataset 2: Segmentation – 2310 instances, 19 features (continuous), 7 classes (balanced)

Each instance in this dataset represents a 3x3 pixel segment of a randomly selected outdoor picture. The features are continuous values containing information about the visual characteristics of the segment. The 7 classes correspond to what is depicted in the segment.

This dataset is interesting because it serves as a stark contrast to the Poker Hands dataset. This dataset has a relatively low number of instances, has more features that are continuous, and contains evenly balanced classes. The expectation is that this dataset will help highlight the differences, strengths, and weakness of various ML algorithms when comparing to the Poker Hands dataset.

Preprocessing

Little preprocessing was required for these datasets. There were no missing attributes to worry about and the number of classes for each was not large enough to warrant feature selection to avoid the curse of dimensionality. On the Segmentation dataset Label Encoding was used to convert classes to integer values. Features were standardized prior to fitting to all classifiers.

Validation

A separate hold-out test set was set aside prior to any implementation. Algorithms were implemented using 5-fold cross-validation to estimate classifier performance on unseen data.

The algorithm scoring used to select the best parameters and to produce the model complexity and learning curves in this analysis is balanced (weighted) accuracy. This is useful in gauging how well an algorithm does on unbalanced datasets such as the Poker Hands dataset where more weight is assigned to classification results of rare classes. This metric is more valuable than a standard unweighted accuracy metric because it allows us to see how the algorithm performs with difficult to classify instances where few training instances have been seen.

2. Decision Trees

The decision tree algorithm was used with Gini Impurity criterion for splitting tree nodes. During trial runs, this criterion generally created trees that yielded better CV scores when compared to information gain although both gave similar results. For this implementation, pre-pruning was used to set restrictions on how a tree can be constructed.

In Figure 1 we can see that at a max tree depth of 12, the decision tree algorithm peaks in test accuracy of the Poker Hands dataset but is continuing to climb in training accuracy. A similar peak can be seen in the Segmentation dataset model complexity curve at a max tree depth of 13 in Figure 3. Beyond these peaks as you lift the restriction of max tree depth you get into the territory of overfitting where the algorithm fits the training data well but performs poorly on unseen test examples. Minimum samples per leaf was found to have minimal impact on accuracy with variance only decreasing slightly in both implementations with rising parameter values due to the algorithm being forced to generalize.

The learning curve in Figure 2 shows a wide spread between training and testing accuracy in the Poker Hands dataset. The algorithm gets better and better at modelling a function to match the training data but the testing accuracy plateaus quickly and at a low accuracy which again is indicative of substantial overfitting and thus high variance. This may be due to the unbalanced nature of the dataset and the fact that the scorer gives high wait to the rare classes. Figure 4 shows that the algorithm performed well with the Segmentation dataset. As training examples increase the variance decreases suggesting more data would yield even better classification performance.

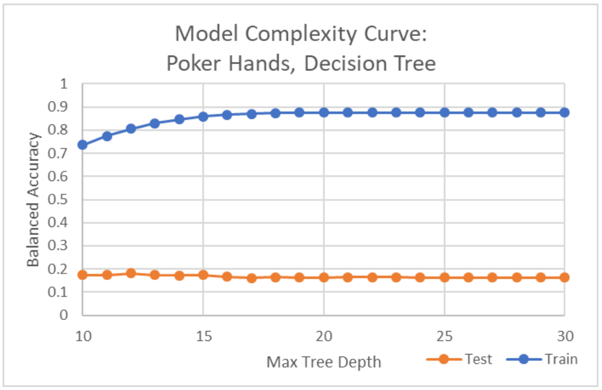

Figure 1

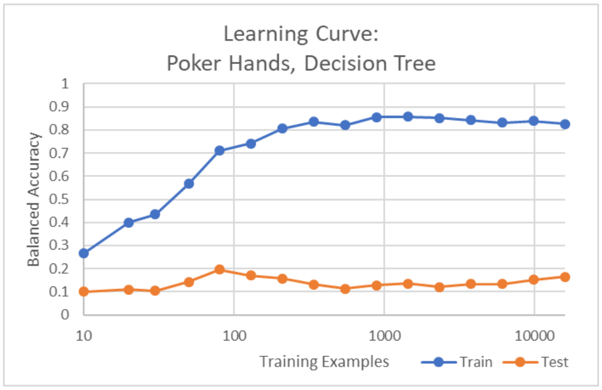

Figure 2

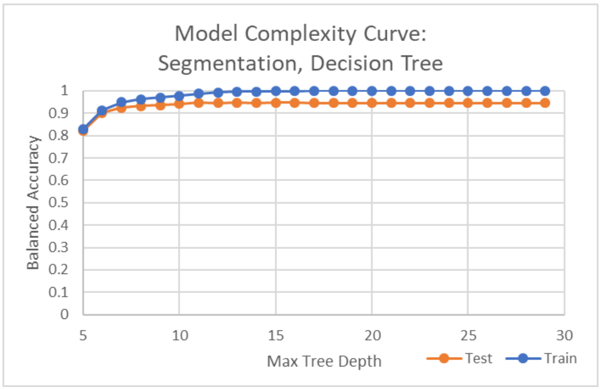

Figure 3

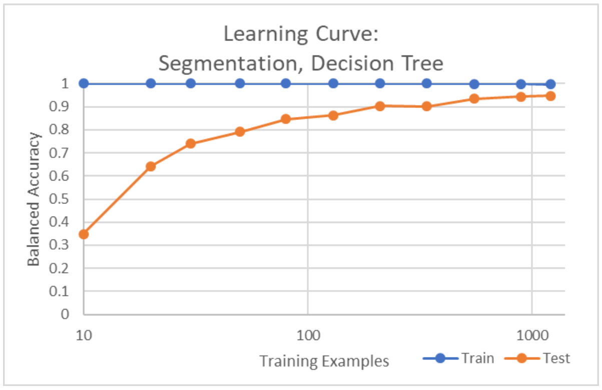

Figure 4

3. Artificial Neural Networks

For the implementation of an Artificial Neural Network, I used the Multi-Layer Perceptron algorithm.

Here we examined the impact of the learning rate and hidden layers to determine the best set of hyperparameters. The learning rate is optimized so that it allows us to reach a minimum error between the calculated outcome and the true class label without overshooting it. In Figure 5 & Figure 7 we determined that the best value of the learning rate is 0.017013. We can also see that in both cases the training accuracy closely meets the test accuracy which suggests this parameter has little variance, however, the Poker Hands dataset does show high bias.

Both implementations seemed to favor higher number of nodes per layer (Figure 6 & Figure 8). Originally the algorithm appeared to be performing poorly on both datasets when compared to others. Based on an early Model Complexity Curve for the Hidden Layers hyperparameter it appeared that I was using too few nodes/layers in my hyperparameter search. This may be because more nodes allows more freedom for the algorithm to create a more complex and accurate model, reducing bias.

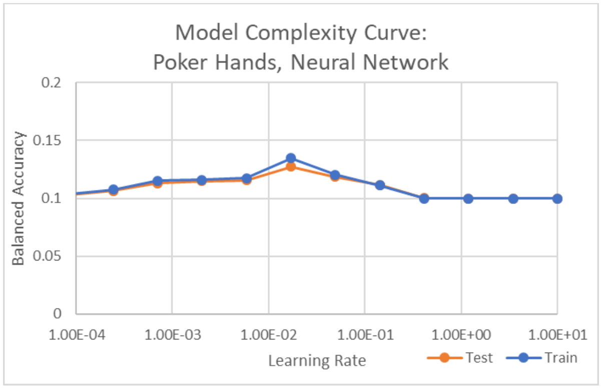

Figure 5

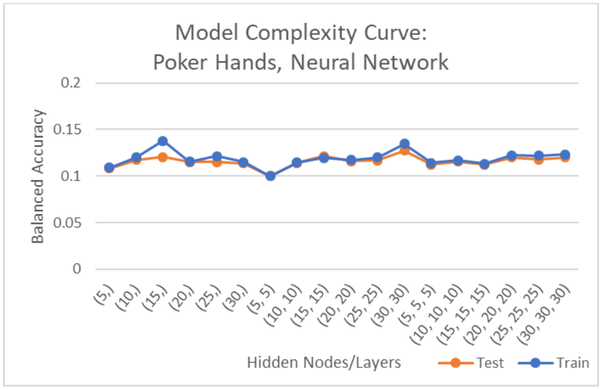

Figure 6

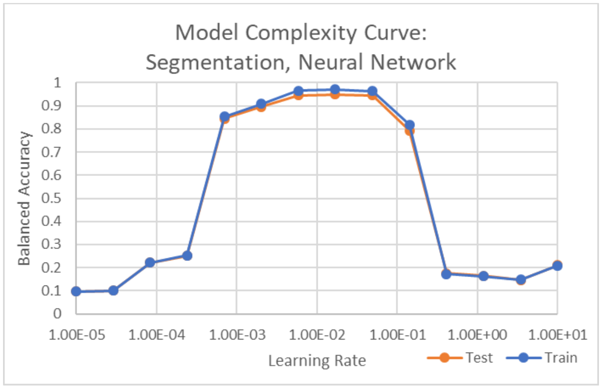

Figure 7

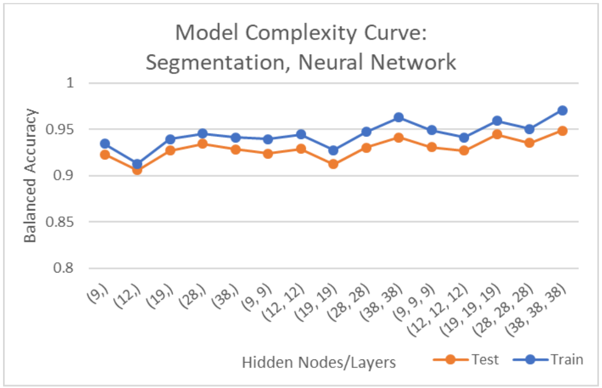

Figure 8

From the learning curves in Figure 9 & Figure 10 we can see there is slight overfitting in both datasets as the number of iterations grow. In the Segmentation learning curve we can also see a drop off in testing accuracy around 20 iterations with further suggests some variance with growing iterations. In comparison to some of the other algorithms seen in this report, the amount of overfitting displayed here is low and we can consider these decent models in that regard.

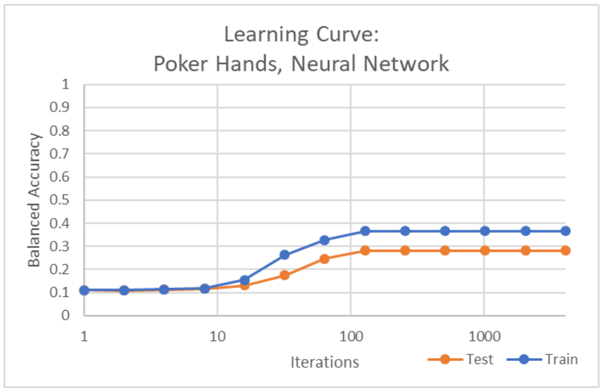

Figure 9

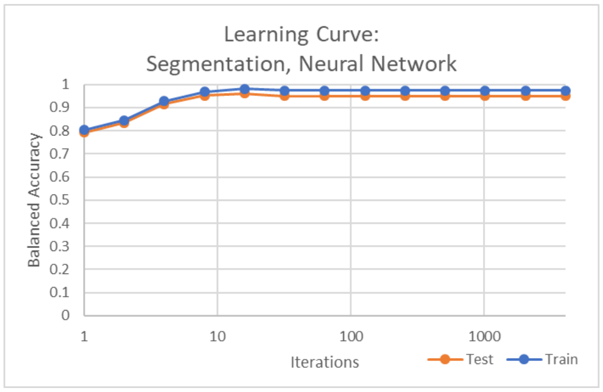

Figure 10

4. Boosting

The boosting algorithm was implemented using AdaBoost. AdaBoost combines multiple weak learners, in this case Decision Trees, to create a strong learner. It does this by implementing each weak learner sequentially with a sample dataset and assigning weights to each instance based on how that learner performed. Instances that were classified poorly are assigned higher weights and are more likely to be in the next training set. In addition, each learner is assigned a weight that determines its role in the strong learner amalgamation based on its performance.

Since the Decision Tree is used as the base estimator, the hyperparameter for max tree depth is examined as well as the number of estimators used in the AdaBoost implementation.

The Poker Hands dataset implementation max tree depth complexity curve in Figure 11 shows high bias and high variance with the peak test accuracy at a max depth setting of 12. This value is the same as the max depth used for the decision tree algorithm. We can see from Figure 13 that the reason the max depth is so high is because the best results were found with only one estimator with variance increasing as number of estimators grows. This seems counterintuitive because we should expect better accuracy with an increasing number of estimators.

To try and gain some insight, another Poker hands learning curve was produced (Figure 14) with the same parameters using an unweighted accuracy score to measure performance. Comparing to the learning curve in Figure 13, which uses balanced accuracy as in the rest of this report, we see that unweighted accuracy increases with the number of estimators but the balanced accuracy does not. This may be because the algorithm is successfully classifying more common classes better and better each time but continually struggles with the rare classes (like Royal Flush) which have a larger effect on the balanced accuracy. Decision stumps were also examined for the Poker Hands dataset with up to 1000 estimators but there is still no improvement over the one estimator case.

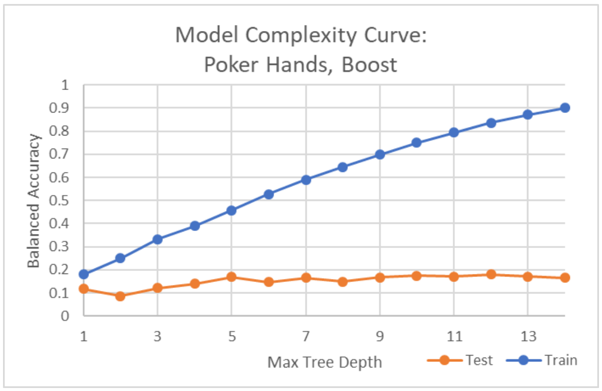

Figure 11

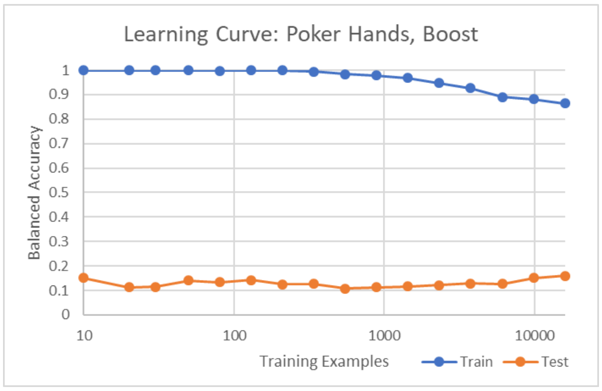

Figure 12

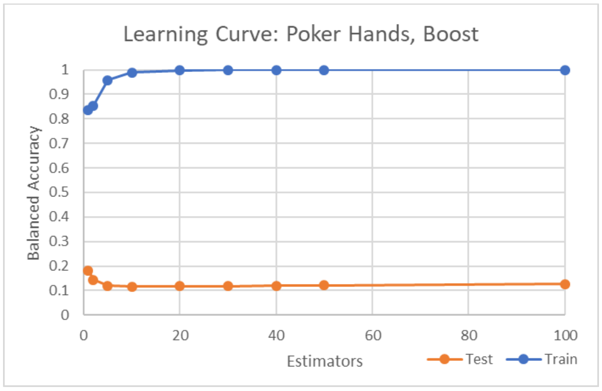

Figure 13

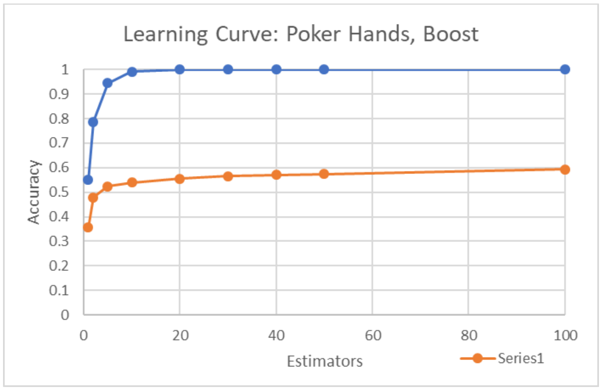

Figure 14

With regards to the Segmentation dataset implementation, we see in Figure 15 that the best max tree depth for the Segmentation implementation is 7. This gives us a superb variance/bias trade off where higher values of max tree depth give us slightly higher variance. Comparing the max tree depth selected here to the Decision Tree implementation max depth of 13, we see that we are much more restrictive in the max depth for the Boost algorithm. This is to be expected as the boost algorithm is comprised of weak learners where the algorithm can be expected to perform well if the learner classifies better than random chance and there are enough estimators being used. Thus more complex decision trees are unnecessary and may overfit the data as we see here for higher max depths.

Figure 16 shows us the best number of estimators for the Segmentation implementation to be 40. This number gives us low variance and low bias. Beyond this number of estimators we can see that there is a slight increase in variance. The boosting algorithm performed very well in accuracy for this dataset in comparison to other algorithms.

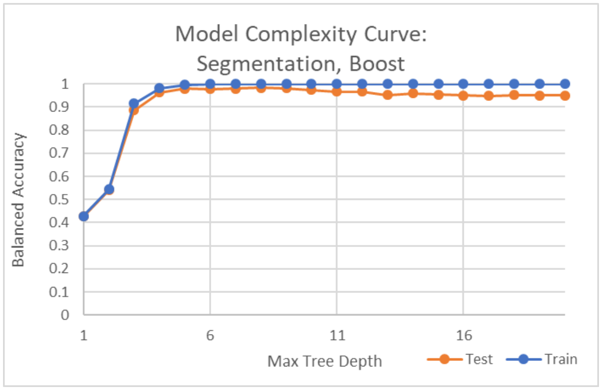

Figure 15

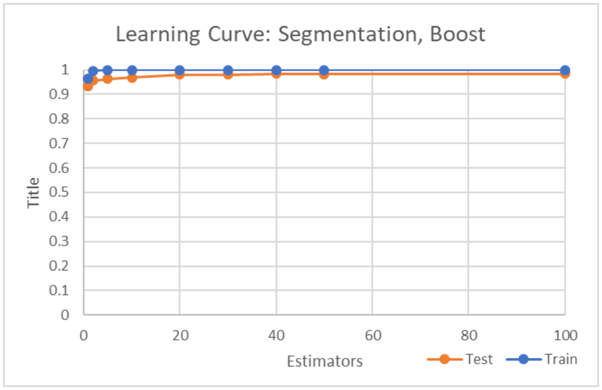

Figure 16

5. K Nearest Neighbors

Hyperparameters for the weights and distance metric were tuned. For both datasets, Distance performed over Uniform for the weight parameter and Manhattan performed over Euclidian for the distance metric.

The main hyperparameter we are concerned with is the value for k. Surprisingly the value for k was found to be best at 1 for the Poker Hands dataset according to Figure 17. This means that the instances in this data set are best classified based on the single closest neighbor's classification. This is likely due to very localized decision regions where neighbors that are close by may belong to different classes. This may be because for this dataset it would be easy to misclassify, for instance, a straight-flush as a flush. In this case you would only want the closest neighbor to influence the classification.

The learning curve in Figure 18 shows high bias and high variance. The curve suggests that with the addition of more training examples we will only see minimal improvements if any. The KNN algorithm performed decently on this algorithm in terms of accuracy. I believe the fact that KNN weights all features in the feature space equally is an asset here. Intuitively since every feature represents a card (either suit or rank) and card order does not matter there should not be any features that are more trivial or noisy than others.

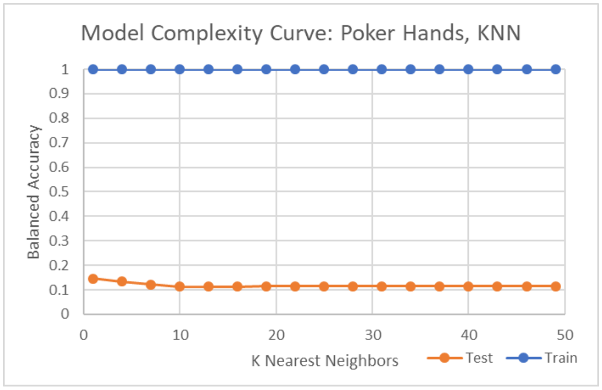

Figure 17

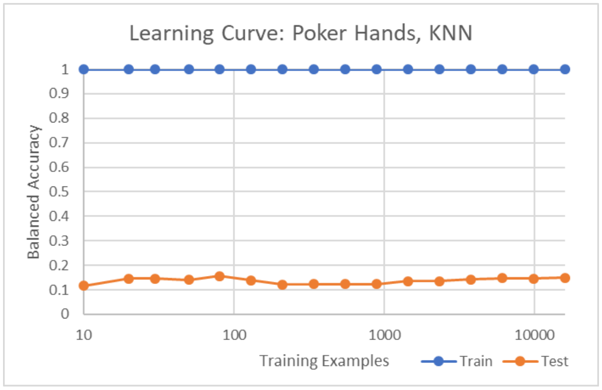

Figure 18

With the Segmentation dataset implementation we also see that testing accuracy generally decreases in Figure 19 with the value of k. The best value of k found is 4 which may be again due to localized and well-defined decision regions where you would not want further neighbors influencing the classification.

The learning curves in Figure 20 show low bias and low variance. The KNN algorithm was one of the best performers on the Segmentation dataset. This may be due to a low amount of noise in the data and it may be that classes are clustered in well-defined regions in the feature space although the low value of k may suggest otherwise.

It is also worth noting that since KNN is an instance based learner that remembers each training example as opposed to adjusting a model based on the classification of a training example, the training accuracy is always going to be 100%.

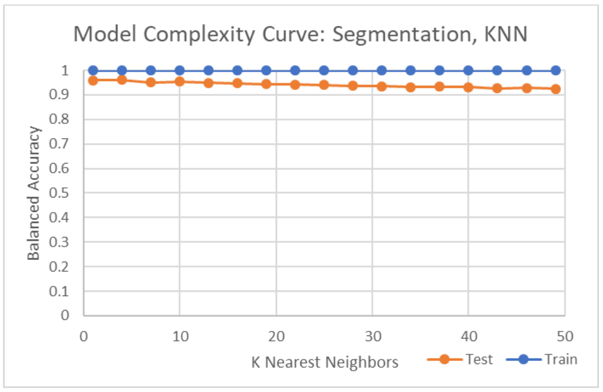

Figure 19

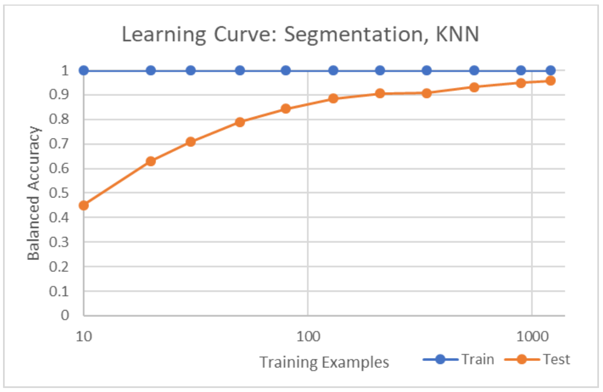

Figure 20

6. Support Vector Machines

Two SVM kernels were implemented for this report: Linear and Radial Basis Function (RBF).

The number of iterations hyperparameter was left unbounded in the linear kernel but capped at 3000 iterations in the RBF kernel in interest of reduced run time. The hyperparameter C was examined using model complexity curve for both kernels. This hyperparameter controls the penalty for misclassification. With lower values for C you will have will less penalty for misclassification errors leading to wider margins whereas with higher values of see you will see smaller margins with a focus on correctly classifying all instances.

In the linear kernel implementation, C was not found to have a great effect on the Poker Hands implementation as shown in the model complexity curve in Figure 21 and we see high bias/error but low variance. The C value of 1000 was the best value found. The value of 10 was selected for C in the Segmentation implementation. Here, in Figure 22, we see much lower bias in comparison to the Poker Hands implementation in addition to also having low variance. The lower value of C suggests that the Segmentation dataset favors a wider margin with lower variance with the tradeoff of misclassifying some instances.

The Poker Hands learning curve shown in Figure 23 demonstrates high bias but low variance as the training and testing accuracies converge but at a high rate of error. However, we see in Figure 24 that as iterations increase we see high variance suggesting that we may be able to attain slightly more accurate classification in lower iterations due to reduction of overfitting.

The Segmentation learning curve shown in Figure 25 shows low variance and low bias with increasing training examples and generally looks to be a good model, however, we will see even lower bias with our RBF kernel. The iterations learning curve Figure 26 also shows us promising results with very low bias and slight variance in higher iterations where the training accuracy reaches nearly 100% while the testing accuracy lags slightly.

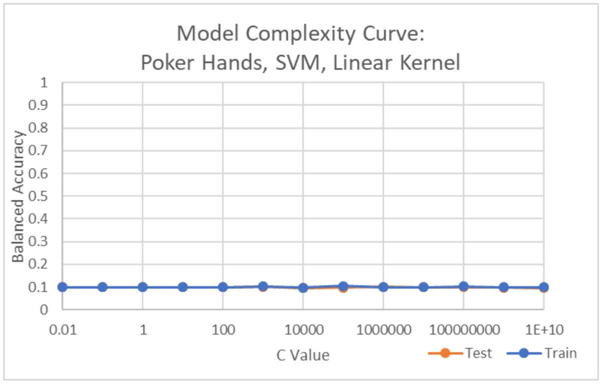

Figure 21

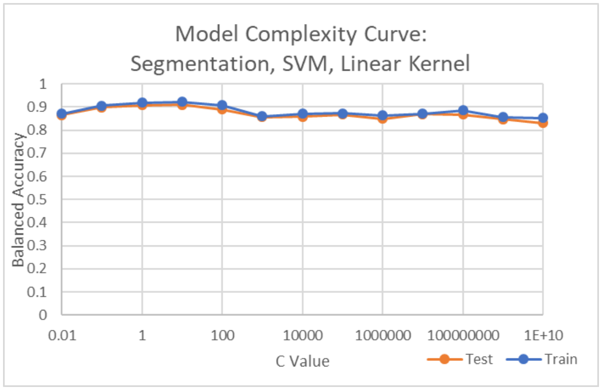

Figure 22

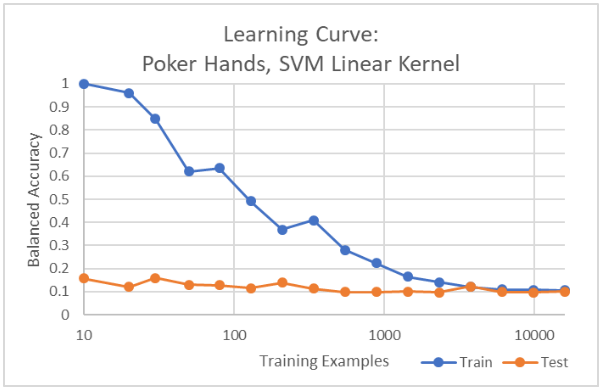

Figure 23

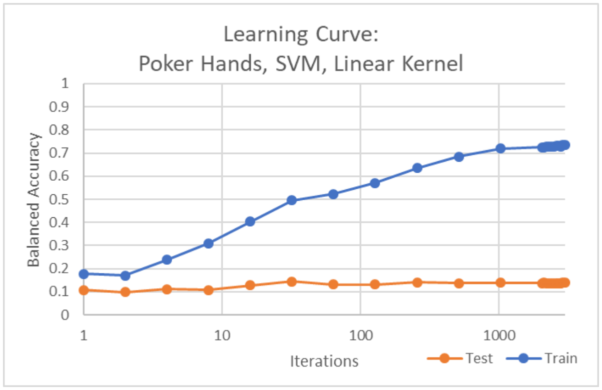

Figure 24

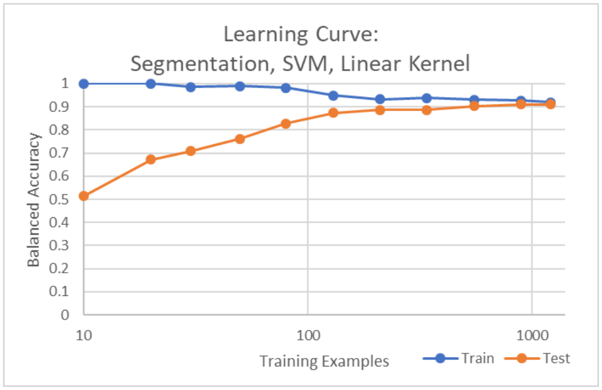

Figure 25

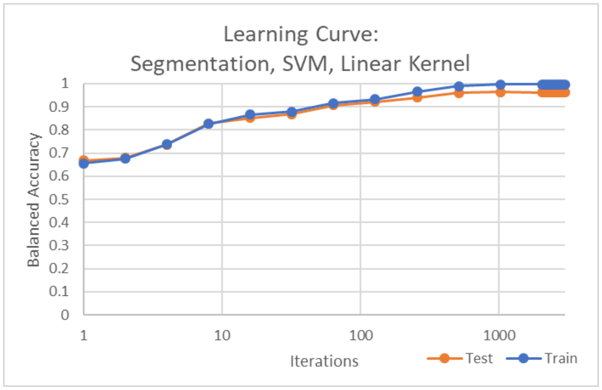

Figure 26

The RBF kernel showed better performance with both datasets when compared to the linear kernel. The Poker Hands model complexity curve in Figure 27 shows us a high variance with increasing values of C. The best value found was 100 as this gave us the highest training accuracy with greater values of C showing higher variance. The model complexity curve for the Segmentation, Figure 28, shows low variance with a peak test accuracy at 1000 with higher values of C showing slightly greater variance/overfitting.

Figure 29 shows high variance in the Poker Hands dataset implementation and a high bias as was shown in our complexity curve. As training examples increase, however you begin to see a reduction in bias and a slight uptick in test accuracy suggesting that more training examples may be beneficial to this classifier. In Figure 30, we also see high variance in the Poker Hands implementation with increasing iterations. Peak test accuracy with the Poker Hands dataset was found to be better in the RBF kernel despite its tendency to overfit.

For the Segmentation implementation, Figure 31 & Figure 32 show excellent results with regards to variance and bias. There is still some slight variance with higher number of training examples and iterations but we can see that the RBF kernel performed better that the linear kernel with lower variance and higher test accuracy.

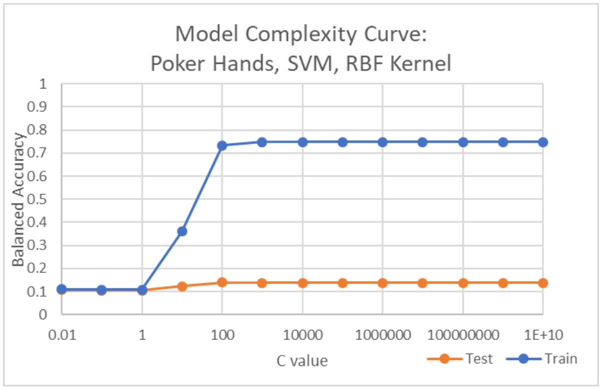

Figure 27

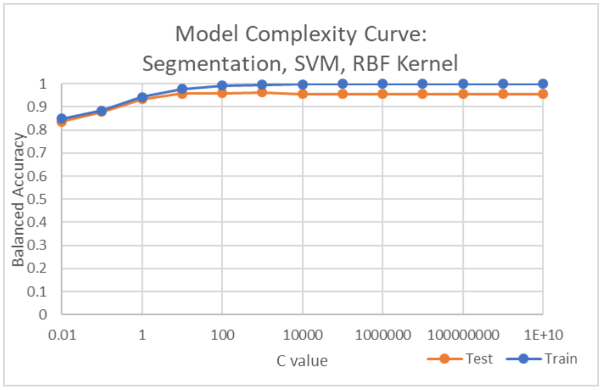

Figure 28

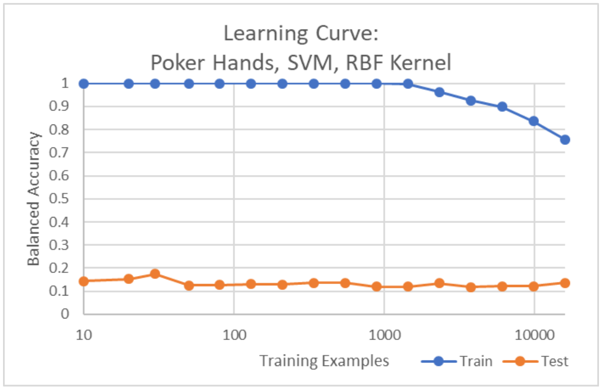

Figure 29

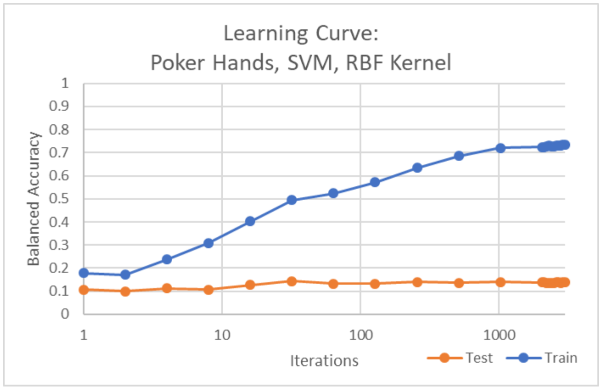

Figure 30

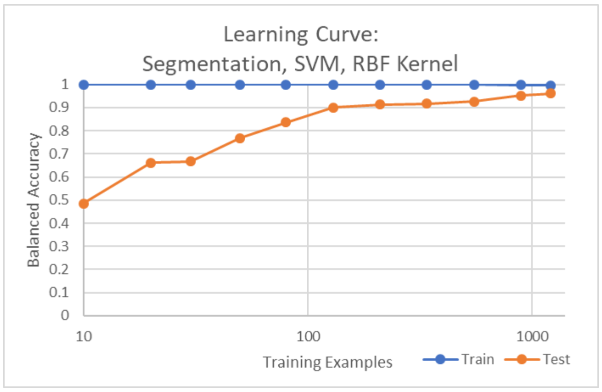

Figure 31

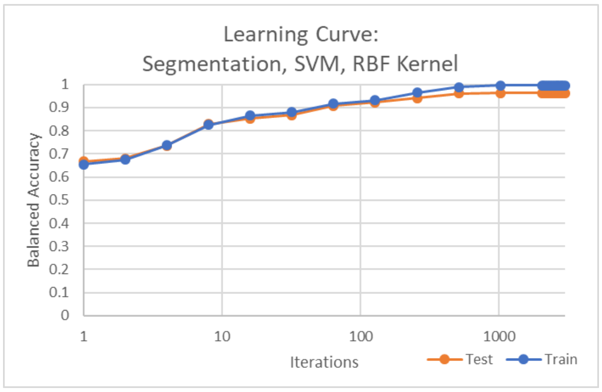

Figure 32

7. Algorithm Comparison

Hold-Out Test Set Accuracy

Figure 33 shows the final performance of each algorithm using the hold out test set for each dataset. The Poker Hands dataset has a separate training set that was used as the hold out set and 30% of the Segmentation dataset was separated out prior to any algorithm implementations. Final testing was performed with these hold out test sets after all algorithm tweaks had been finalized.

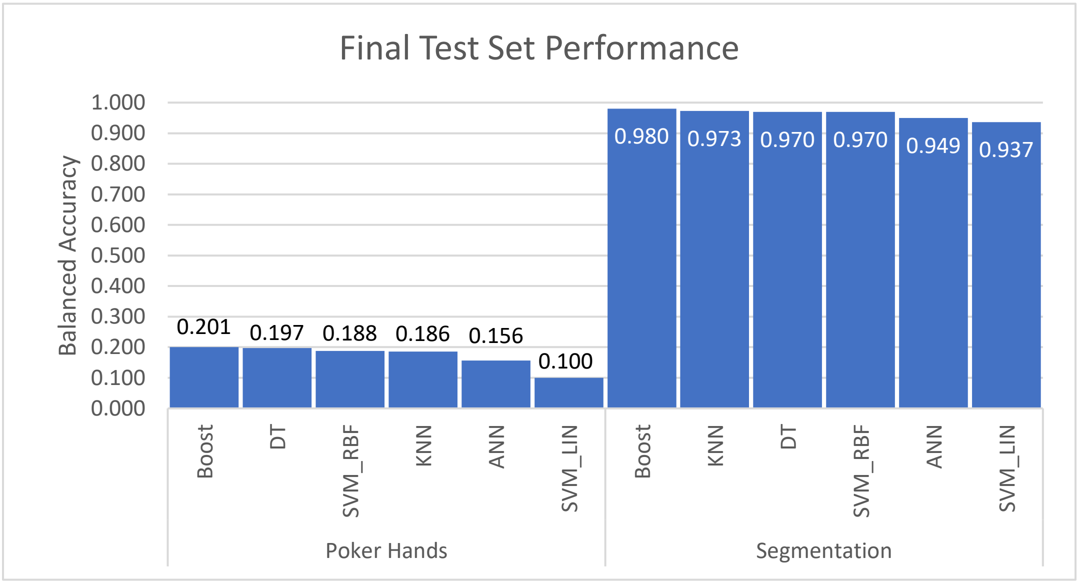

Figure 33

The results of the algorithm testing for the Poker Hands dataset were interesting. The Boost algorithm (which incorporates only one Decision Tree estimator) and the Decision Tree algorithm performed best. One possible explanation is that the Poker Hands dataset has very little noise. The algorithms that performed best tend to do poorly with noise in general (Boost can handle noise well but with more estimators) but excel in datasets with low noise. It may be a situation where the poorly performing algorithms generalized a bit too much and accounted for noise that wasn't there.

The Segmentation dataset also was saw best results with boosting. The algorithm appears to handle balanced datasets very well and with more estimators noise can be accounted for easily. The tradeoff is a long run time as we will see in our timing curves.

Another interesting result is that the linear kernel for SVM gave us the lowest accuracy of any algorithm tested here for both datasets. We can see from the learning curves in our earlier analysis that the bias is high relative to other algorithms. This algorithm may be struggling to model the target function with enough complexity due to inductive bias. We see that the RBF Kernel gives us much better results with both datasets likely due to decreased inductive bias but the tradeoff is a much slower test time due to a quadratic runtime complexity.

In general, Segmentation dataset was more easily classified by all algorithms. One of the main reasons for this is probably the fact that this dataset is evenly balanced while the poker dataset is unbalanced to an extreme degree. Throughout this analysis we have been grading our algorithms on balanced accuracy and it is clear from constructing confusion matrices for KNN and Linear SVM that some algorithms are struggling to classify the rare Poker Hands classes. Another likely reason is that the Segmentation classes may be divided into more clearly defined decision regions without as much overlap. This seems to be suggested by the KNN implementation where the best value for k on the Poker Hands implementation was found to be 1 versus 4 in the Segmentation implementation.

Timing Curves

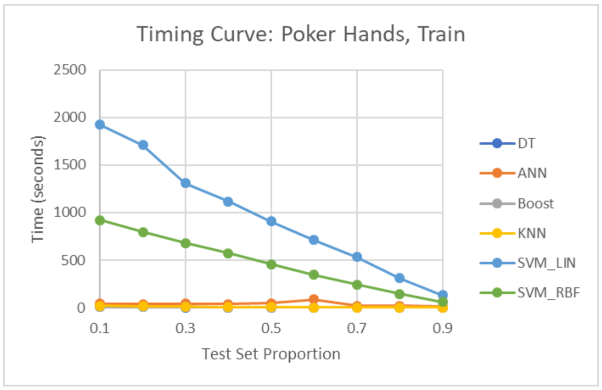

Figure 34

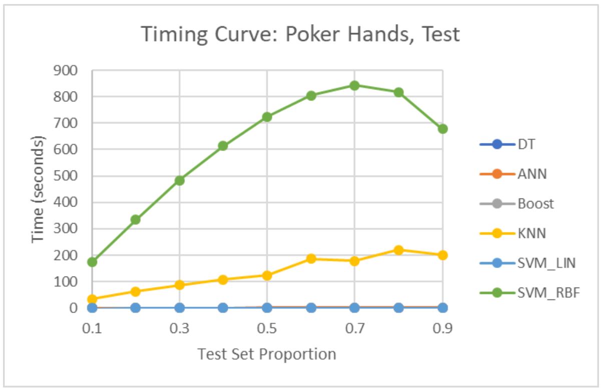

Figure 35

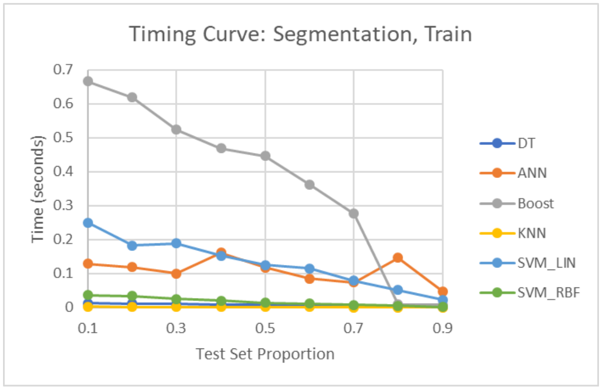

Figure 36

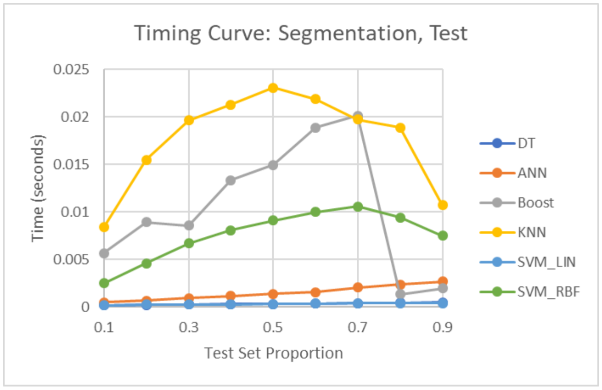

Figure 37

Timing curves were generated for the both datasets on each algorithm with best parameters. Timing curves for training and testing were created with respect to increasing test proportion size. What we see is that Neural Networks and Decision Trees have very short runtimes for both datasets for both training and testing relative to other algorithms. These are eager learners however and we can see that training does take more time than testing.

KNN, on the other hand, due to being an instance based lazy learner, we can see that training runtime is very quick but you can see in Figure 35 and especially Figure 37 that testing time is much more longer. You can see the curse of dimensionality is also coming into play with the KNN implementation on the Segmentation dataset due to the higher number of features in comparison to the Poker Hands dataset.

Boosting, as an ensemble learner, understandably has a high training time and high testing time. We unfortunately do not get to see that in the Poker Hands implementation due to only one estimator being used. However in the Segmentation implementation we see in the Training Timing Curve, Figure 36, that boosting gives us the longest training time with increasing training examples giving us a mostly linear curve. This is due to the number of estimators, with each estimator constructing a 7-depth decision tree. Testing time in Figure 37 is large as well due to the algorithm weighting each training instance following testing to prepare for the next estimator as well as weighting all estimators based on test performance.

SVMs, being eager learners, had a long run time in training as seen in Figure 34 in runtime with larger training sets. The Poker Hands Linear SVM training took quite a long time but this is made up for in test run time. In Figure 35 & Figure 37 we see that linear SVM has a trivial test time but the RBF kernels test time is substantial with growing test sets. It appears that the test runtime complexity for the RBF kernel is dependent on the training set size with test runtime decreasing with much lower training set sizes. This may be due to a reduced number of support vectors.

Conclusion

It appears that for both datasets, the Decision Tree algorithm is the optimal classifier that strikes a good balance between testing accuracy and run time.

For the Poker Hands dataset it is disappointing that better performance was not achieved and is a testament to how difficult it is to model a training set with very low occurrences of certain classes and easy to misclassify classes. The decision tree dataset seemed to deal with these issues best – to the point where the boosting algorithm was found to give best results with only one decision tree estimator.

The Segmentation dataset was best modeled by the Boosting dataset but the run time was significant. The second-best implementation, KNN, also showed quite a large runtime in testing. The Decision Tree algorithm, while not quite as successful at classifying (97.0% test accuracy vs. 98.0% boosting accuracy), had a relatively quick runtime. I think the low runtime makes up for the slight loss in classification accuracy, making Decision Trees the best performer.