**Supervised Learning**

smatthews39@gatech.edu

Sean Matthews

In this assignment you will find a detailed analysis of different supervised learning algorithms. We will look at different parameters and analyze the corresponding algorithms. The learning curves provided will be analyzed with the timing data and the maximum iterative parameters. Below is an overview of the data used.

*Data chosen*

For the Supervised Learning project, I wanted to choose an interesting subject to me, although it may not fare well to others, and I wanted to see how learners can make a difference in fantasy baseball. My data sets aren't applicable to a draft, or to picking up individuals in the form of transactions, but I wanted to get an insight on how machine learning can predict how well a player would do in his career. My two data sets are listed below in greater detail.

I began with wanting to look at how one may be able to predict if a player would ultimately be inducted into the Hall of Fame. However, with only ~250 players ever being inducted, my data set ended up being extremely unbalanced and the analysis was telling me that there was not enough data in one of the classes to properly create graphs, and visual representations of the learners. This led me to the data I picked. I chose to gather the history of baseball and every stat that has ever been captured from the year 1890 to the year 2017 and create a pivot table within excel that gathered every one of the 19,103 players career stats. I removed some stats that seemed a lot more weighted to the stats I was capturing, and ultimately ended with 15 stats for the batting set, and 12 for the pitching set

*Batting Set*

For the batting set, I began with wanting to look at how one may be able to predict if a player would ultimately be inducted into the hall of fame. However, with only ~250 players ever being inducted, my data set ended up being extremely unbalanced and the code was actually telling me that there was not enough data in one of the classes to properly create graphs, and visual representations of the learners. This led me to the data I picked. I chose to modify the history of baseball and every stat that has ever been captured from the year 1890 to the year 2017 and create a pivot table within excel that gathered every one of the 19,103 players career stats and predict how many runs that player would score. To be fair to some players, such as pitchers who showed very poor performance in hitting and helped the learner greatly, I modified the set to only include those who had 400 or more at bats. I wanted to predict the RunRating that I chose for this dataset, Table 1 shows the ratings.

| Career Runs | 0-100 | 101-300 | 301-500 | 501-700 | >700 |
|---|---|---|---|---|---|
| RunRating | 0 | 1 | 2 | 3 | 4 |

**Table 1:** RunRating Distribution

*Pitching Set*

The pitching set was a bit more straight forward when coming up with what I would want to predict. One of the most common stats in pitching is Earned Run Average (ERA). This is calculated by:

ERA = 9 x Earned Runs Allowed / Innings Pitched

An ERA that is sub 4.0, is considered good, but one that is below a 3.5 are the favored pitchers. The sub 3 level, are those who will most likely be on ballots to be inducted into the hall of fame. For my analysis I am using 3.5 as my classifying line, with a minimum innings pitched of 166.7, or 500 outs.

| > 3.5 | < 3.5 |
|---|---|
| 0 | 1 |

**Table 2:** Binary Classification for ERA

## Decision Tree

For my decision tree, I pruned to a max depth of 5 initially but also wanted to investigate what a max depth of 8 would do to the graphs, and below are the results of this experiment. Pruning generally decreases training set accuracy but assists in achieving better accuracy on test data. As well as looking at the learning curves for this classification, I found it informative that the best parameter was the statistic 'Hits'. This showed that the more hits a player had, it was directly correspondent to the RunRating. Anyone familiar with baseball would know that if a player can get on base, they have a higher chance of scoring, and this is used in the mathematical approach to baseball called sabermetrics, which is widely used throughout Major League Baseball. Two of the most commonly used stats are 'Hits' and 'On Base Percentage'. For the pitching statistics, the best parameter was 'Shutouts'. A shutout is where a pitcher pitches the entire game, 9 innings, without giving up a single run. Statistically this has both sides of the fraction for ERA; 1) It has a 0 on the numerator, and 9 on the denominator, which ultimately is a 0, strengthening someone's case for a sub 3.5 ERA.

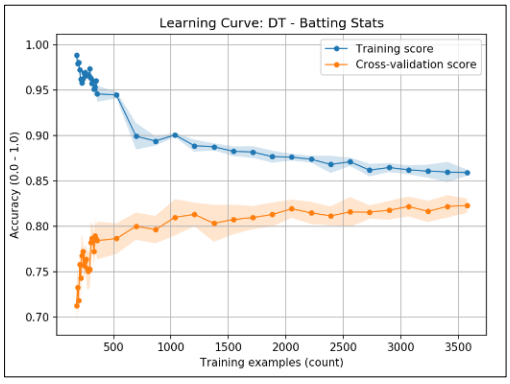



**Figure 1:** DT Classifier Max Depth 5 (Batting)

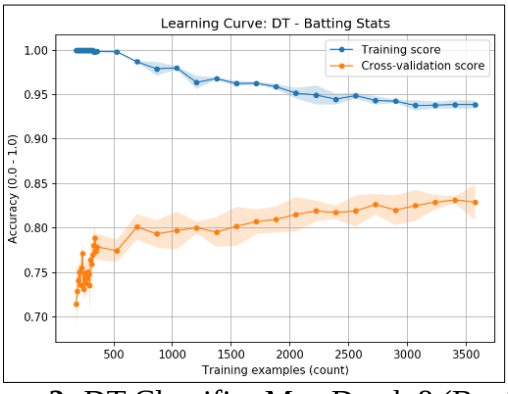



**Figure 2:** DT Classifier Max Depth 8 (Batting)

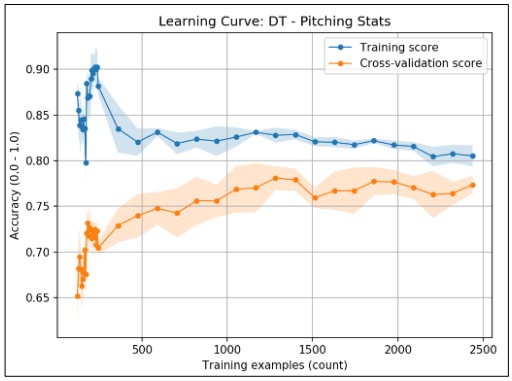



**Figure 3:** DT Classifier Max Depth 5 (Pitching)

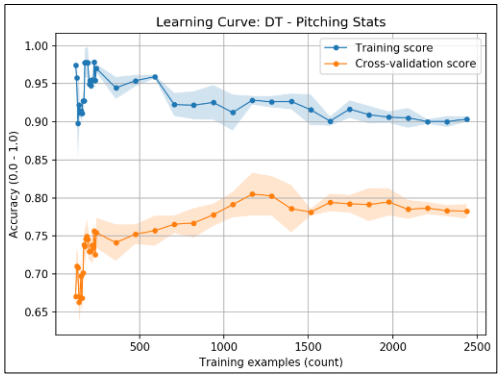
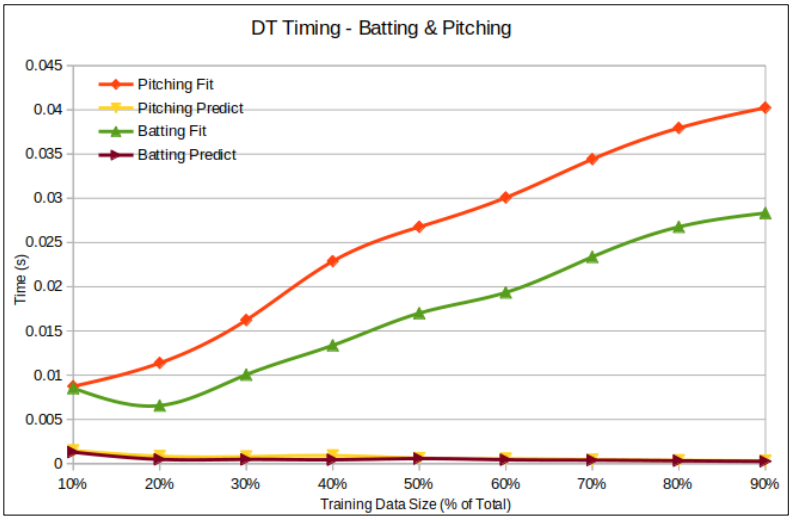



**Figure 4:** DT Classifier Max Depth 8 (Pitching)

The learning curves can help distinguish what is actually happening with the data and the model. With just about each of these curves, the training score starts off as overfit to the training examples, but as more and more examples are added the overfit is reduced on the training set and improves the cross-validation (CV) score. Overfit models normally possess high variance. The graphs that could benefit from more examples would be the two batting curves, as the training and CV set both look as if they




have a bit more room to converge. The pitching curves wouldn't benefit from more data. They too began pretty overfit with their training score. It's also worth noting that with the max depth being set, there is a significant benefit with setting a lower number to be the max depth as it reduces bias and reduces variance.

**Figure 5:** Decision Tree Train and Test Timings

Later on, in the report you will see other timing graphs that are different the one above. The decision tree timing graph has the fit data taking an increasing long time as the dataset size increases, while the prediction set decreases with time, nearly unnoticeably though. As more rows were calculated, more trees are found to add another classifier to the tree.

## Neural Networks

For neural networks, the parameters that were considered were hidden layers, activation function and the initial learning rate. Given these, if our dataset is complex, then a higher number of hidden layers should be used. However, this could cause overfitting and bad performance if it is used on a simple

problem. The activation function determines the behavior of the neurons in the model which also plays a significant role of the performance dependent on the dataset. My graphs showed extreme precision among the training and the CV sets. It had a relatively high biased, but when comprehending the graphs, it shows that as training examples become more plentiful, the accuracy of both the training score and the CV score go up. Through the first 600 training example iterations on the pitch dataset, the ANN only produced an accuracy for both of 50%, but with 2000 examples, the accuracy improved to approximately 95%, meaning our model is learning, and it isn't overfitting.

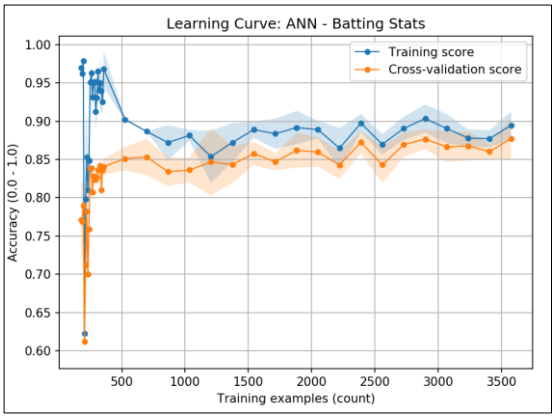



**Figure 6:** ANN Learning Curve (Batting)

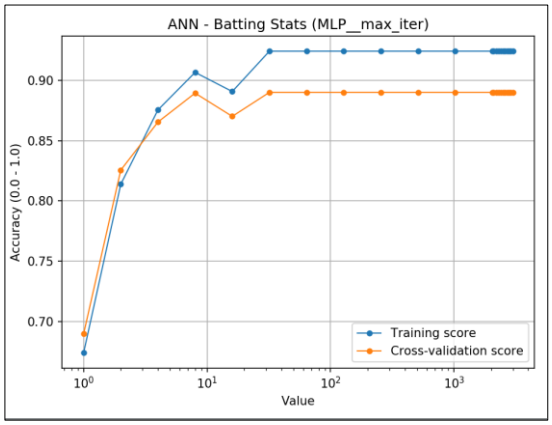



**Figure 7:** ANN Iteration Curve (Batting)

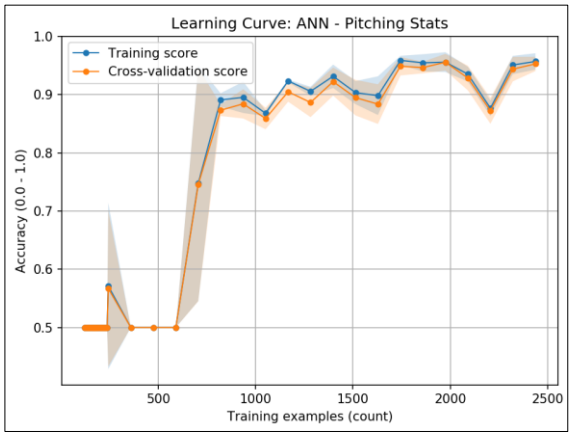



**Figure 8:** ANN Learning Curve (Pitching)

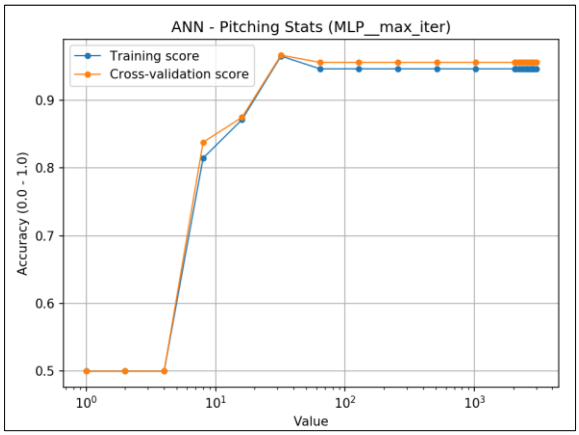



**Figure 9:** ANN Iteration Curve (Pitching)

For our batting dataset, we see a more common graph where the model doesn't quite begin learning until about 500 examples. Before then, there is high variance, and possibly some overfitting for some of the sets of examples. After 500-1000 examples the model doesn't benefit from more examples. In the max iteration graph for the pitching stats, Figure 9, it shows that the CV score is slightly better than the training score, this is a very rare occurrence, but does happen and just shows that the model did a good job at learning with the training score and well fit the data on the CV score. The ideal graph is where the training score and the CV score are nearly identical but just at a high accuracy, not a perfect accuracy, this would signify overfitting. As you can see in Figure 8, the learning curve for the pitching stats is nearly 90% accurate.

## Boosting

Decision Trees are inherently a weak learner and boosting or adaboost can significantly boost the decision tree performance. It does this through iterations. Interesting enough, in Figure 10, the training set is nearly perfect through all amounts of training data for both, batting and pitching stats. Unlike a decision tree, unpruned other than setting a max depth, boosting uses a confidence limit to maximize or minimize the pruning it demonstrates. To analyze Figure 10, the training set is shown to be too simple for pruning, but the CV set does a good job at improving itself over larger sample sizes. Training set signifies extreme overfitting, however the model itself doesn't. I would have expected either a very good score or the exact opposite but given that the model showed on small data sizes only, poor performance, it quickly leaves the realm of overfitting and learns quite significantly. Both of the boost learning curves would benefit from more training examples.

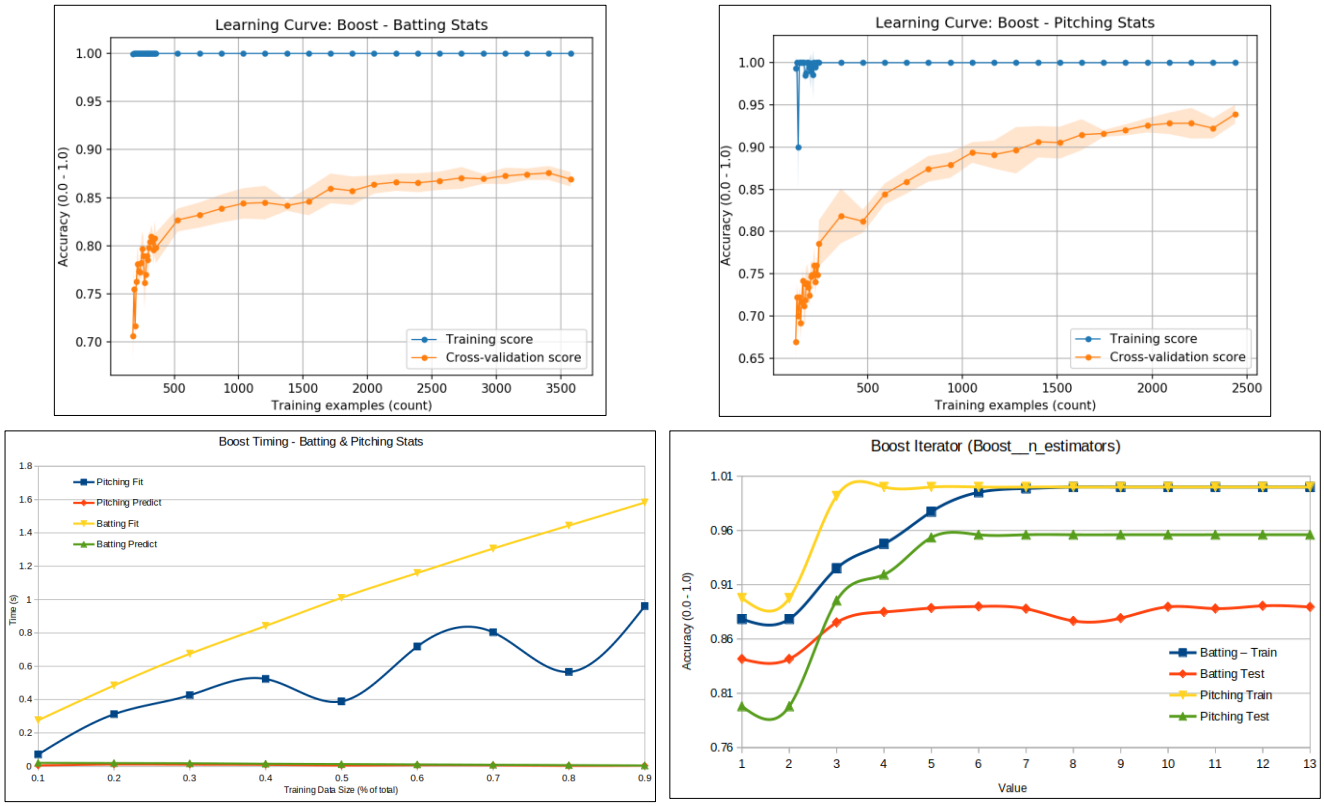



**Figure 10:** Boost Batting (TL), Boost Pitching (TR), Boost Timing Batting & Pitching (BL), Boost Iterator (BR)

In relation to the timing, again, the training took a very small time to complete while the test timing took much longer. It actually took over a second to train the batting dataset as well as almost a second to train the pitching dataset. As decision trees are easy to prune with max depth, boosting allows there to be a tradeoff between bias, variance as it is known to overfit. Giving more control to the algorithm allows it to be easier to tune in the case of pruning. The Boost Iterator graph shows how

## k-Nearest Neighbors

With kNN, my graphs stand out a little bit to me due to that my training score also increases in the beginning. This is most likely attributed to the model only seeing the nearest neighbor for each of the features, when in reality baseball statistics can be very sporadic. In example, a pitcher may have low strike outs, but a lot of Earned Runs, this may signify that the pitcher was really not a good player, but in reality, he was a ground ball pitcher who played 15 years. Bartolo Colon has been playing for 20 years and has racked up great stats including strike outs and complete games, but has a 4.12 ERA. This occurs quite frequently and with KNN, that node being examined may be looking at numerous pitchers like Bartolo Colon and will therefore fail more often. Unlike Decision Trees, KNN uses those around, to predict the classification. In Figure 11 and 12, both the pitching and batting stats follow this trend of a lower training score, but with more examples, they reduce error and become better learners.

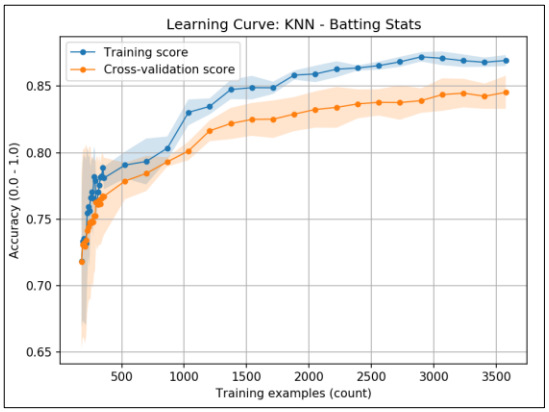



**Figure 11:** K-Nearest Neighbor (Batting)

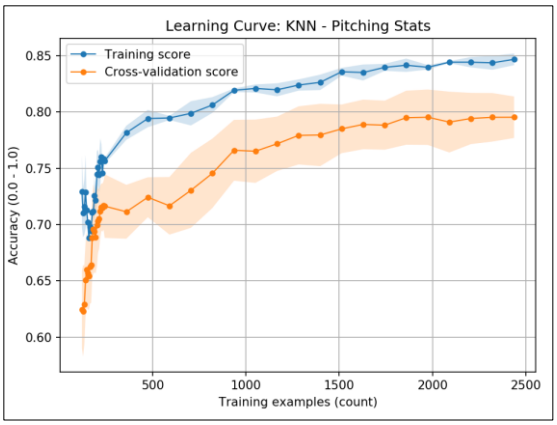



**Figure 12:** K-Nearest Neighbor (Pitching)

Both of these models do not have a high variance and suffer slightly from underfitting. As the example increase, the variance stays low, and the bias stays the same, however, the performance of the model increases significantly.

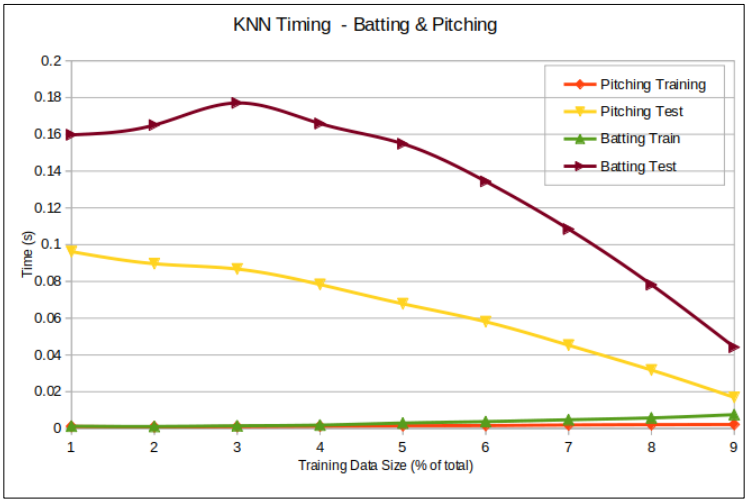



**Figure 13:** K-Nearest Neighbor Train and Test Timings

For the timing of KNN, I compressed the two timings onto a single graph for simplicity and to minimize report clutter. Both data sets performed nearly identical for the training sets, but the test sets

took significantly longer until the larger datasets were utilized. In example, at 3, or 30% of the dataset, the training set for batting stats took 0.001453 seconds, while it too 0.1771 seconds to fit the data. That is more than 100 times the time that it took for the training data. When the full data set is used, we see that the timing decreases drastically and the fit timing decreases to 0.044, which is significantly better than the slowest time of 0.1771 seconds. I imagine with a larger dataset, the nearest neighbors would be more deterministic of the queried item, where with smaller data, it would have harder time being able to classify a neighbor since there are not enough data points.

## Support Vector Machine

Support Vector Machine (SVM) is known as a discriminative classifier that attempts to generalize the data by separating the classes by lines. If the data can be separated by a line, it attempts to maximize the distance of the line, or hyper-plane, between both classes. With each feature we add, we essentially added an additional dimension in the solver trying to separate these classes with hyper-planes. With the code I 'stole', I implemented 2 types of SVM kernels; a radial basis function (RBK) and a Linear kernel.

As a visual representation of how well the SVM learner did on both data sets, the normalized confusion matrices for both the Linear and RBF SVM are posted below.

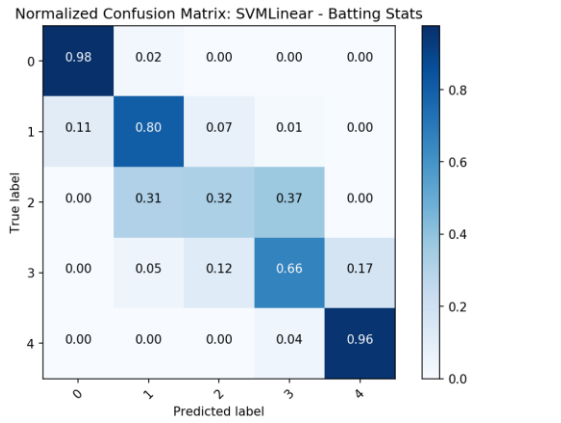



**Figure 14:** Confusion Matrix Linear (Batting)

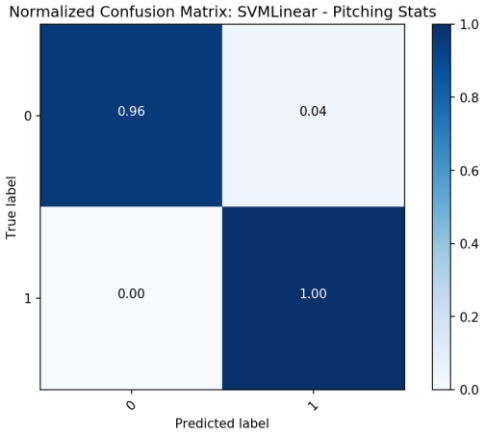



**Figure 15:** Confusion Matrix Linear (Pitching)

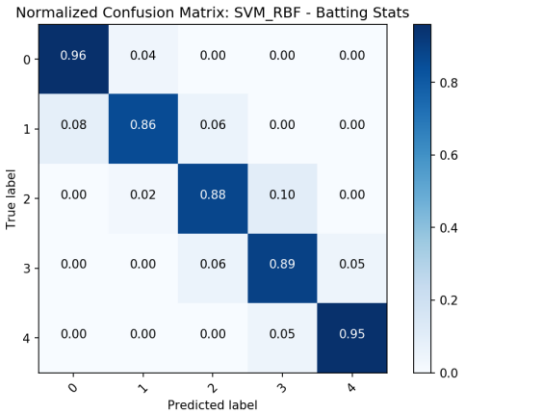



**Figure 16:** Confusion Matrix RBF (Batting)

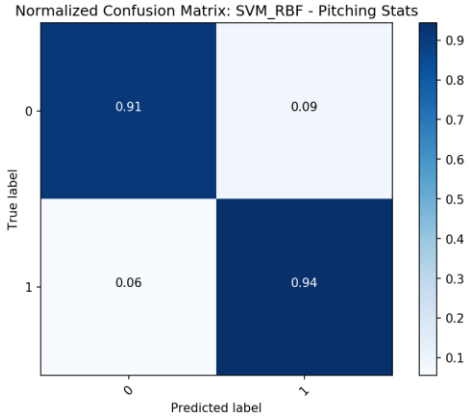



**Figure 17:** Confusion Matrix RBF (Pitching)

**Batting Data Set**

With the Normalized Confusion Matrices, the Linear SVM had a difficult time learning on the batting stats. I would attribute this to there being more classifications and more dimensions, making it nearly impossible to simply draw a line separating the classes.

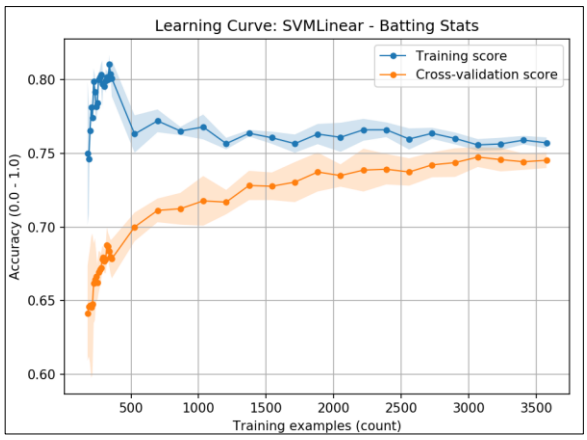



**Figure 18:** SVM Linear

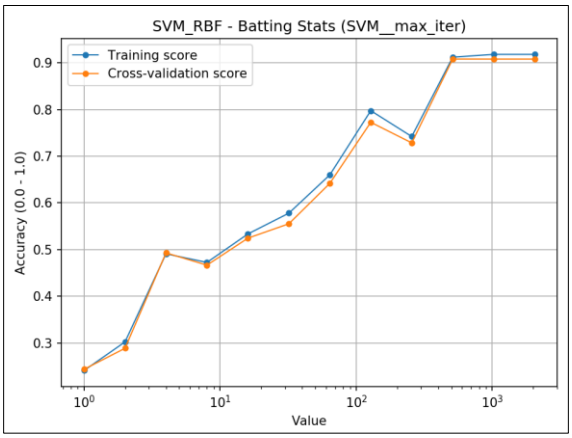



**Figure 19:** SVM RBG Max Iteration

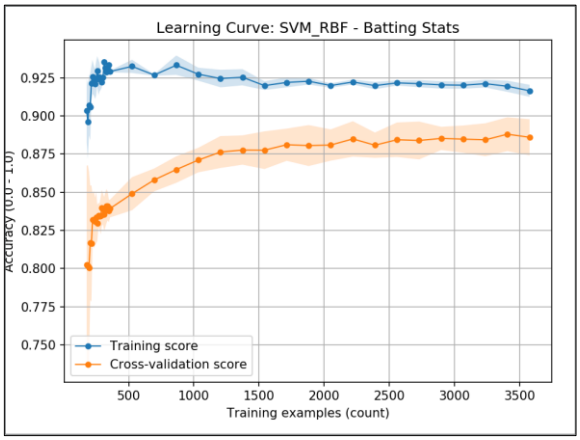



**Figure 20:** SVM RBF Learning Curve

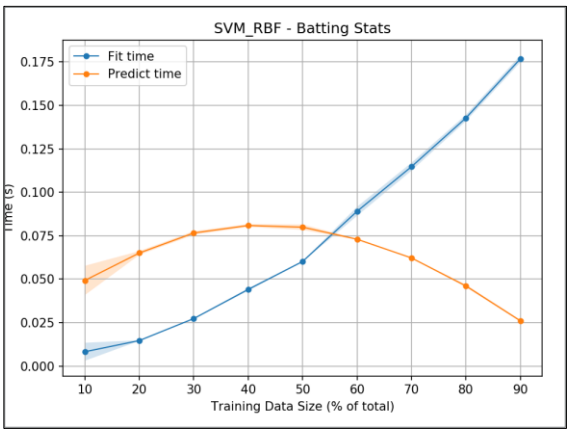



**Figure 21:** SVM RBF Timing

Above shows 4 graphs from the SVM Learner. The top left is the only graph provided from the linear kernel. I found it interesting that between the confusion matrix, and the related learning curve, it is noticable that the model didn't not perform well at all. It had a high amount of variance and overfitting. The accuracy model shows approximately 75% accurate, and this shows in the confusion matrix where those with a RunRating of 2 were rather difficult to predict.

The RBF model shows a high variance and overfitting as well. As the number of training examples increase for RBF, the model learns until about 1500 examples and then quickly flattens. Adding more training examples wouldn't do any good for this model. The lines look to be approaching their horizontal asymptotes, so they will most likely not converge. The model gets better over more examples, and the overfitting is reduced significantly from ~200 samples to 1500 samples.
The RBF kernel of SVM shows that as the iterative parameter increases the accuracy of the model increases. Once the iteration reaches a magnitude of 3, it peaks at its max accuracy of approximately just more than 90%. In the RBF timing data, it shows that more and more time was taken to fit the data, while the predict time decreases. This could be due to the model has too much knowledge, that it is unable to take a fully educated prediction on the classification so it just begins taking guesses.

**Pitching Data Set**

The learning curve for SVM was, to me, surprisingly more ideal than others. With the training set starting with a low accuracy the model tries to fit everything for both the training and the cross validation (CV) set. As the examples increase the model learns and increases its accuracy and begins generalizing. With more examples, the training and CV line begin converging signifying that variance is decreasing. Near ~2500 samples, the lines converge.

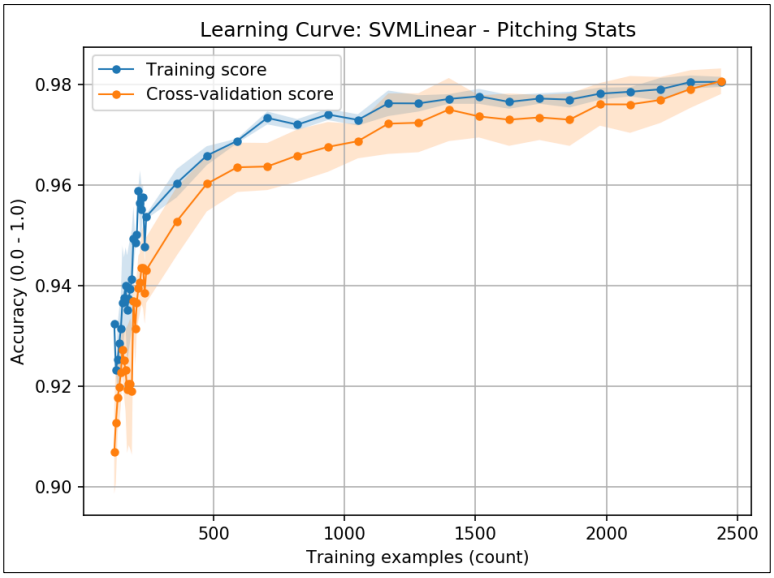



**Figure 22:** SVM Linear Learning Curve (Pitching)

The timing graph and the iteration graphs are found in Figure 23-25. The timing graphs illustrate that as the data size increases, the time it takes to train the data also increases, however, the time to predict a value is relatively similar throughout the data size, other than the downward trend at the end of the graph for RBF kernel. There are multiple reasons why it takes an abundant amount of time to train SVM, the first being the use of RBF. This is a very complex kernel, and the difference can be noted by

the y axis on both of the timing graphs (linear & RBF). The linear kernel had a max time of about 0.036 seconds to train 90% of the data, however, the RBF took upwards of 0.15 seconds, nearly 5 times more than linear. Another parameter that plays a big part in the train time is the data size. The big O notation for SVM is O(n^3). When comparing SVM timing to that of KNN, it is interesting to see that testing KNN takes approximately the same time as SVM takes to train.

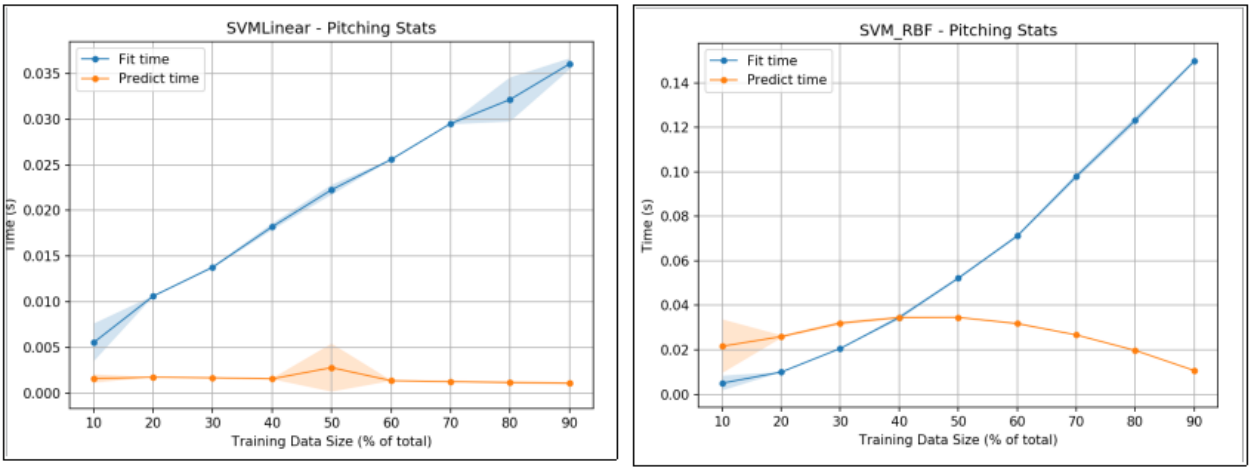



**Figure 23:** Timing Graphs for Pitching Stats (Linear & RBF)

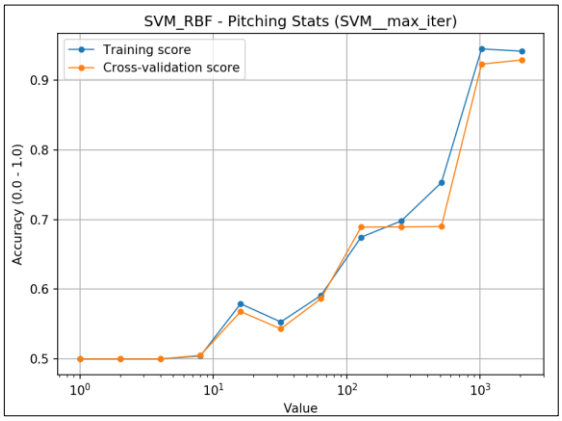



**Figure 24:** Max Iteration for Pitching RBF

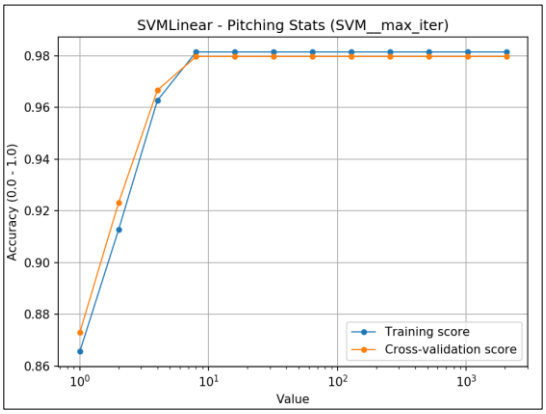



**Figure 25:** Max Iteration for Pitching Linear

The iterations are found to be helpful for RBF kernel as it struggles to obtain a decent accuracy until the parameters are iterated over $10^3$ times. With a linear kernel, the maximum iterations needed before the model achieves its best accuracy is less than 10. Overall the linear kernel seemed to fend better in the analysis done against the RBF for dataset. The training time, maximum iterations, and the learning curves all should that the linear kernel for SVM outperformed the RBF.

**Conclusion**

After the above analysis, attempting to summarize the results seems impossible, but I will break it down by datasets. I believe the batting data and the pitching data played the largest difference in the

findings. With the batting data, we are trying to compare batters that have 5000 at bats to those who have 500 at bats and compare the amount of runs they have. There are a lot of players who are able hit well, and played a long time, but didn't score that many runs. This may have made it difficult on the learners as it may have been harder to classify each. I think this is most relevant in the linear SVM model for the batting (Figure 18). Those who were trying to be classified as 1, 2 or 3 were a lot less accurate than those classified as a 0 or a 4. I do believe that the performance was greatly relatable to the data I chose, as I think looking at the wine data set, would have been a little easier to classify as it looks at predetermines numbers and chooses the classification, while stats is a different with how a player performs. Baseball is known as the sport of numbers, and you are considered a good player if you succeed 30% of the time. This may play a part in the performance greatly. On the other hand, the pitching stats may come across pretty straight forward and if you don't have relatively decent stats, and a healthy arm, you most likely won't have a long career. Now the minimum innings pitched that I filtered was only 167 innings, which most players can accumulate in a year if they are in the starting rotation. I think another factor in my data is the fact that there are closing pitchers and starting pitchers.

To say which algorithm performed best there are different factors, but judging off of my analysis, I actually believe I'm most impressed with the KNN scores. Although the test timing scores show that it took a longer time to predict a classification, I like seeing that both the training score and the CV score increased as more data samples were added. I shows that it was underfit originally but moves to becoming quite a good fit. To say who the best is with iterations, I would say Linear SVM performed the best, as it was able to find a trend within 10 iterations of the parameters.

I did use cross validation for all of my algorithms and I think it greatly helped. Iterating through different sets of data to test and train seemed to give a bigger and better picture of the model it was being tested on. If I were to make changes to the algorithms, I would try different depths for the decision tree, change some of the pruning parameters for adaboost. I'd be most interested in seeing how changing my data and adding a $6^{th}$ level of classification to the batting data and possible a sub 3 ERA classification to my pitching data.

Big thanks to Chad Maron, who allowed me and hundreds others to steal his code.

References

Cmaron. “Cmaron/CS-7641-Assignments.” *GitHub*, 8 Feb. 2019, github.com/cmaron/CS-7641-assignments/tree/master/assignment1.

“Learning Curve.” *Ritchieng.github.io*, www.ritchieng.com/machinelearning-learning-curve/.

Mitchell, Tom M. *Machine Learning*. McGraw Hill, 2017.

Olteanu, Alex. “Learning Curves for Machine Learning.” *Dataquest*, Dataquest, 21 Jan. 2019, www.dataquest.io/blog/learning-curves-machine-learning/.