Predicting the Outcomes of Baseball Games Using Machine Learning Approaches

Mike Delevan

December 4, 2019

1 Abstract

Major League Baseball, also known as America's pastime, continues to be one of the most popular sports in the US and contains a faithful fanbase that closely pays attention to the statistics surrounding the game. This study intended to capture the interest in the statistics of baseball by utilizing openly avaiable data to train machine learning algorithms to predict the outcome of upcoming games. To achieve the goals of this study the data was first programmatically loaded, pre-processed, visualized, and feature engineered to transform it into acceptable input for the machine learning algorithms. Then, multiple machine learning algorithms were researched and tested to determine the architectures that performed the highest. The chosen models were then evaluated for their accuracy using mutliple methods.

Conclusions need to go here when I actually get conclusions

2 Problem Description and Objectives

Major League Baseball is one of the oldest and most popular sports in the US and thus has a wide range of data available for statistical and predictive analysis. This data can be found in a wide range of places on the internet and can contain information on just about every occurrence in any baseball game right down to the pitch. Using this data, a lot can be predicted such as a pitchers next pitch, the probability that a team will win their next game, or even who is most likely to win the World Series. This project will pull this historical data from online databases and use it to predict the probability that a team will win their next game and the margin of victory. A recent attempt at predicting the winner of a certain matchup using machine learning and artificial neural networks achieved a precision of 58%**(Cite)**. Given the task, this precision is actually very good because it is hard to accurately predict the winner of a game due to the amount of variables. If data scientists were able to predict the winner of baseball games with 90% accuracy sports betting would most likely cease to exist. The goal of this project intends to utilize exploratory data analysis to achieve a greater than 50% precision on predicting the winner of upcoming baseball games.

The objectives for the system will be to take historical and recent baseball game data and use it to predict the winner of upcoming baseball games and the game's score. It will also need to be represented graphically so it is easy for a user to see the predictions for upcoming games. The data will need to be programmatically loaded, pre-processed, and analyzed to find patterns within the data for feature engineering. Various neural networks and machine learning algorithms will be researched, designed, trained and tested to find the best architecture that will best be able to predict the outcomes and scores of games. This may include trying several different models to see which one performs the best and mitigating any underperforming neural network architectures. The predictions from the chosen architecture will then be evaluated using multiple evaluation methods.

3 Data Description

The dataset chosen for this project will be sourced from Retrosheet and include baseball data from 2010-2018. Retrosheets has a large amount of historical baseball data ranging from 2018 all the way back to 1871.

It includes downloadable play-by-play files, complete game logs, and team schedules for all of the years mentioned above. Retrosheets also has multiple regular season game log files compiled into a zip archive that can be downloaded in bulk. This project will be using those compiled game files for seasons from 2010 through 2018. The initial format of the data is a simple text file, however all fields are in comma delimited format so conversion to a CSV file will not cause a problem. These files contain the most complete and most relevant data to solve the problem and include information such as team statistics, winning and losing pitchers, linescores, attendance, starting pitchers, umpires and more. The initial form of the data does not include headers for each field, so these will have to be added manually. In total the dataset has close to 46,000 rows and 161 variables that will need to be processed, below is a detailed listing of all variables:

Fields(s)	**Description**
1	Date as a string in the form "yyyymmdd".
2	Number of the game corresponding to the current season.
3	Day of the week as a string.
4-5	Name and league of the visitor team.
6	Game number of the visitor team.
7-8	Name and league of the home team.
9	Game number of the home team.
10-11	Runs of the visitor and home team, respectively.
12	Length of game in outs. A full 9-inning game would have a 54 in this field. If the home team won without batting in the bottom of the ninth, this field would contain a 51.
13	Day/night indicator ("D" or "N").
14	Completion information indicates if the game was completed at a later date (either due to a suspension or an upheld protest).
15	Forfeit information.
16	Protest information.
17	Park identifier.
18	Attendance.
19	Duration of the game (in minutes).
20-21	Visitor and home line scores as a string. For example, "010000(10)0x" indicates a game where the home team scored a run in the second inning, ten in the seventh and didn't bat in the bottom of the ninth.
22-38	Offensive statistics of the visitor team: at-bats, hits, doubles, triples, homeruns, RBI, sacrifice hits, sacrifice flies, hit-by-pitch, walks, intentional walks, strikeouts, stolen bases, caught stealing, grounded into double plays, awarded first on catcher's interference and left on base (in this order).
39-43	Pitching statistics of the visitor team: pitchers used, individual earned runs, team earned runs, wild pitches and balks (in this order).
44-49	Defensive statistics of the visitor team: putouts, assists, errors, passed balls, double plays and triple plays (in this order).
50-66	Offensive statistics of the home team.
67-71	Pitching statistics of the home team.
72-77	Defensive statistics of the home team.
78-79	Home plate umpire identifier and name.
80-81	First base umpire identifier and name.
82-83	Second base umpire identifier and name.
84-85	Third base umpire identifier and name.
86-87	Left field umpire identifier and name.
88-89	Right field umpire identifier and name.
90-91	Manager of the visitor team identifier and name.
92-93	Manager of the home team identifier and name.
94-95	Winning pitcher identifier and name.
96-97	Losing pitcher identifier and name.
98-99	Saving pitcher identifier and name.

Fields(s)	Description
100-101	Game Winning RBI batter identifier and name.
102-103	Visitor starting pitcher identifier and name.
104-105	Home starting pitcher identifier and name.
106-132	Visitor starting players identifier, name and defensive position, listed in the order (1-9) they appeared in the batting order.
133-159	Home starting players identifier, name and defensive position listed in the order (1-9) they appeared in the batting order.
160	Additional information.
161	Acquisition information.

4 Exploratory Data Analysis

Loading and Formatting Data

Initially, Java was used to pre-process the data but it was later discovered that this would be more work than if I did it in Python. The Java program first took in the raw data year-by-year and created csv files with headers for each team's data in that year. The decision to use Java was made because I thought I would later use it for web scraping. However, after some consideration I decided to do my entire project in Python instead of Java, Python, and C#. The raw data can be downloaded from Retrosheet. This project utilizes the game logs from the 2010-2018 seasons. Since the Java program split the data into year by year files for each team, it would be necessary to combine them back into one file for one team. This could've been done with only one file using Python but because I split it into multiple files, I am restricted in the initial loading of the data.

After the data was combined and each team had their own file with their own respective data it was time to pre-process the data before use in a machine learning model. This included converting names to be consistent across all files, removing nulls and categorical variables, calculating meaningful statistics, and condensing the dataset. The first issue that was noted in the pre-processing phase was that the Marlins changed their name from the Florida Marlins to the Miami Marlins sometime during the 2012 season. The problem with this is that it also changed their abbreviation in the dataset, which meant that there was two abbreviations for the same team. To fix this error, each team file was accessed and anytime a MIA abbreviation was seen it was converted to FLO. This made sure that each file had a consistent abbreviation for the Marlins across all files so that it can be referenced correctly in the future.

The next step for data pre-processing was to calculate some important sabermetric values for each team. This step is called feature engineering because new features will be created using features already found in the dataset. The important statistics that were calculated include batting average (BA), on-base percentage (OBP), slugging percentage (SLG), on-base plus slugging percentage (OPS), and if the team won the game. Below is the formula for each of these statistics:

$$BA = \frac{Hits(H)}{At - Bats(AB)}$$

$$OBP = \frac{Hits\ (H)\ +\ Walks\ (BB)\ +\ Hit - by - Pitch\ (HBP)}{At - Bats\ (AB)\ +\ Walks\ (BB)\ +\ Hit - by - Pitch\ (HBP)\ +\ SacFlys\ (SF)}$$

$$SLG = \frac{Singles\ (1B)\ +\ 2\ *\ Doubles\ (2B)\ +\ 3\ *\ Triples\ (3B)\ +\ 4\ *\ Homeruns\ (HR)}{At - Bats\ (AB)}$$

$$OPS = On\ Base\ Percentage\ (OBP)\ +\ Slugging\ Percentage\ (SLG)$$

All of these were done effortlessly and quickly using the Pandas package in Python. Using Pandas, we are able to calculate values for an entire column in one line of code instead of having to iterate through 40,000 rows of data one by one and calculate the value manually. These new variables were calculated for both the home team and away team regardless which team file we were accessing. To make sure that the values calculated and the formulas used were correct, Baseball-Reference was used to verify a small subset of these calculations.

Another interesting thing happened during this step, I got a warning saying 'TypeError: unsupported operand type(s) for -: 'int' and 'str" which I thought was strange. After some investigation into my files I found that some lines had information that was offset by about 3 cells. Below is a snippet of what this offset looks like. What happens is that a game will get suspended and restarted on a different day. Retrosheet documents this restart by adding in 3 new indeces into their data, thus throwing off my Java parser.

This small error had only a few occurences in the dataset, about 35 bad rows out of about 40,000. It was solved not programmatically but rather manually because doing it programmatically would haven taken up too much development time. After this issue was solved it was time to do feature subset selection.

Feature subset selection intends to reduce the size of a given dataset so that it only includes the most important variables. In this case, each row was distilled down to a record of statistics for the team mentioned in the file name. For example, the data found in the NYAFinal.csv would only contain data for the Yankees instead of having data for both the home team and away team. Many of the original 75 variables were removed in this step such as balks and triple plays because they were found to be useless in modeling. At the end of this feature subset selection, there were 22 variables left all in the perspective of the team mentioned in the file name. Again this was done simply using Pandas, however in this case the modification of copying of data had to be done row by row. This was because it had to be determined whether the data we were modifying and copying over was from the home team or the away team's perspective. Below is a list of the features that were selected for the rest of the EDA and modeling phase of the project.

Feature	**Description**
teamAbbr	Abbreviation of the team mentioned in the name of the file
League	League (AL/NL)
Score	Score from the given game
isHomeTeam	Whether the team mentioned in the file was the home team in a game
atBats	Number of at-bats in the game
Hits	Number of hits acheived in the game, includes home-runs
Doubles	Number of doubles acheived in the game
Triples	Number of triples achieved in the game
homeRuns	Number of home runs achieved in the game
RBI	Number of runs batted in during the game
Walks	Number of offensive walks in the game
Strikeouts	Number of offensive strikeouts in the game
LOB	Number of runners left on base in the game
pitchersUsed	Number of pitchers used in the game
indER	Number of individual earned runs achieved by a pitcher
teamER	Number of team earned runs achieved by all pitchers
Errors	Number of defensive errors committed
battingAverage	Batting average achieved in the game
OBP	On-Base-Percentage achieved in the game
Slugging	Slugging Percentage achieved in the game
OPS	On-Base-Plus-Slugging Percentage acheived in the game
Win	Whether the team mentioned in filename actually won the game

One last step that needed to be completed before moving on was calculating 10 game averages for each team for prediction purposes. This step is needed for predicting upcoming games, which is the intended

functionality of the system. When predicting upcoming games, there will be no data from that game to predict off of, therefore we must create it by some means. This can be done through a built in function in Pandas called describe(). The describe fucntion will return a list of statistics for each numerical variable in the dataset. The most recent 10 games was fed into this function and the mean for each variable was extracted and placed into an averages file with all other team averages. To truly test the system, this 10 game average will be fed into the trained models to predict upcoming games in the 2019 season.

Data Visualization

Now that the data has been pre-processed, the data can then be visualized to its fulles potential. To make the visualization more effective, all of the pre-processed data was combined into a single file so that it includes data for all team, which equates to roughly 30,000 lines. The goals of the visualization aspect of the exploratory data analysis are to determine variables that may be good indicators of a win, determine how those variables differentiate teams, and validate our findings using a correlation matrix.

First, the summary stats were collected for each variable to look for any preliminary quirks and features in the dataset. One important finding was that in the dataset there was approximately 50% rows with a win value and approximately 50% rows with a loss value. This is important for machine learning because we want to give the model evenly distributed data. Training on an even distibution of win and losses might even reduce overfitting because the model will be generalized to both win and loss. If there were more wins in the dataset, the system would learn from that pattern in the data and may end up predicting more wins than losses, which is not the intended function of the system. Another finding from the description is that on average, teams score around 4 points a game which may seem low, but compared to soccer it is a pretty good value. Also, on average a team can attribute at least one run to a homerun and the other 3 points through batting in base runners.

To find the relationships between all variables in the dataset, a scatter matrix was created. This visualization shows scatter plots between each variable and a distbution plot for each varaible as well. Because this plot has every variable combination in it, we will omit this plot and instead present specific plots that were selected from the scatter matrix. Most of the plots related to the win column are bar plots because there are only two values in that column. The first graph that will be discussed is the graph depicting the relationship between Win and OPS.

OPS vs. Win

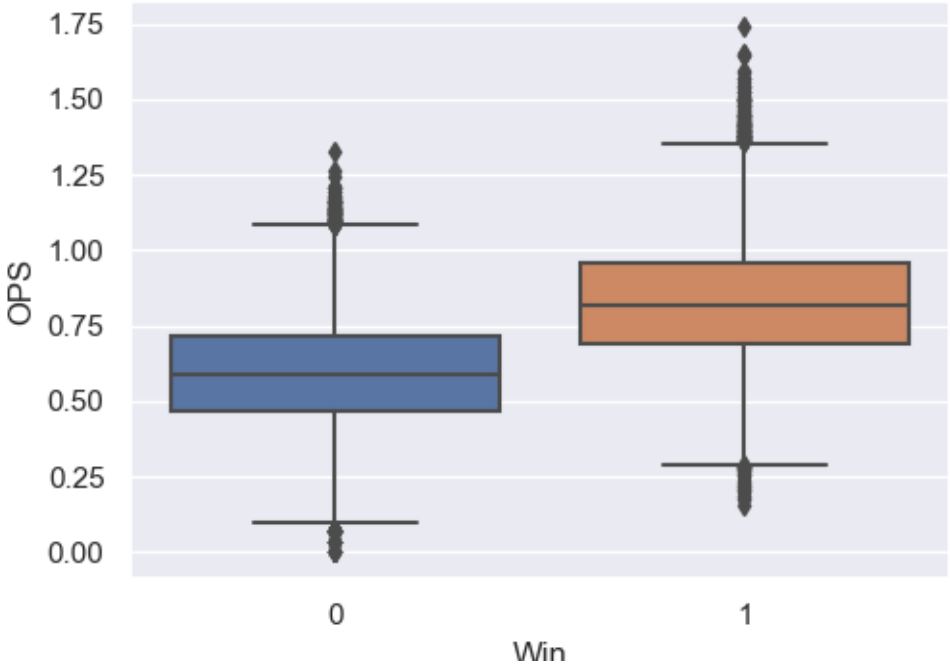


Figure 1: Graph depicting relationship between OPS and Win

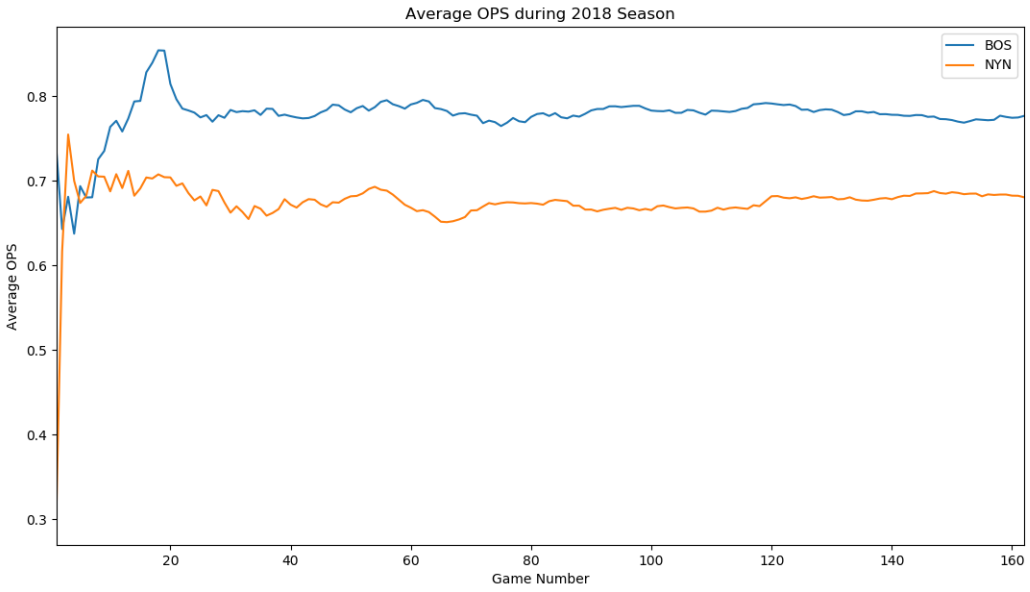


Figure 2: Difference in OPS values in 2018 between Red Sox (Blue) and New York Mets (Orange)

Based on the graph in Figure 2, it was inferred that the higher a team's OPS value is, the greater the chance they have of winning a game. This means that when it is fed into a machine learning algorithm it may be a good indicator of a win. The OPS value is a combination of both slugging percentage and on-base percentage and is typically an indicator of offensive prowess. This means that the higher the OPS value, the better a team is offensively, thus leading to more hits and runs. However, just having this figure is not enough to confirm our assumption that OPS is a good predictor for a win. To confirm our suspicions the Pearson's correlation coefficient value was calculated. The correlation values can range from +1 to -1, where +1 indicates a perfect positive correlation while -1 indicates a perfect negative correlation. If the correlation value falls between ±.50 and ±.99, then there is a strong correlation between the variables. For this relationship, a correlation value of **.524** was calculated which is relatively high for correlation values. Because of the high value, it was determined that there is a strong positive correlation between OPS and Win. This finding agrees with our previous assumption that a team's OPS is a high indicator of a win. It's not a direct correlation, but there is some kind of relationship between OPS and Win.

To further investigate the difference that OPS can make on win predictions, the average OPS for each team over the 2018 season was plotted and is shown in Figure 3. This figure illustrates the average OPS for both the Red Sox and the Mets over the 2018 season. The figure shows that over the course of 1 season, both teams converged to two different average OPS values, with the Red Sox having the higher OPS value and the Mets having the lower OPS value. Using our previous finding that OPS is a good predictor of a win/loss, we could infer that the Red Sox would win more often than the Mets. After checking the Baseball-Reference standings, it was discovered that the Red Sox won a humongous 108 games during the 2018 season versus the Mets' measly 77 wins. Again this confirms our inference that a higher OPS can lead to a better win percentage.

Strikeouts vs OPS

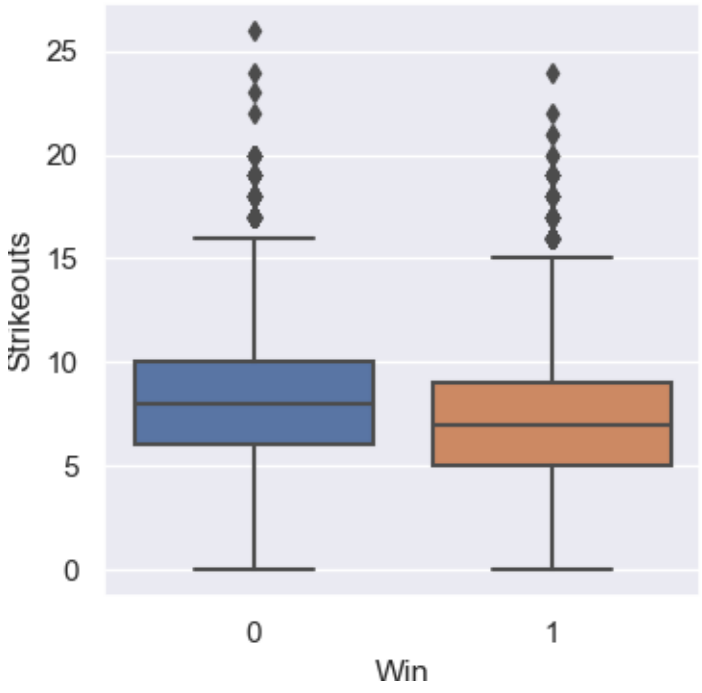


Figure 3: Graph depicting relationship between Strikeouts and Win

The next relationship that was investigated was between Win and offensive strikeouts. Based on the graph in Figure 4, it was inferred that there was a very small negative correlation between strikeouts and win. As seen in the figure, the mean strikeouts for both win and loss are relatively close with a large amount of outliers at higher values. The graph in the figure could indicate that strikeouts are not a very strong predictor of a win or loss because of the narrow difference in the means. The correlation value calculated for this relationship was **-.156** which is relatively close to 0. This value tells us that there is a small negative correlation between strikeouts and wins. This means that strikeouts really are not a good indicator of a win/loss and may need to be removed to improve performance if needed. Although there is a slight negative correlation between these two variables.

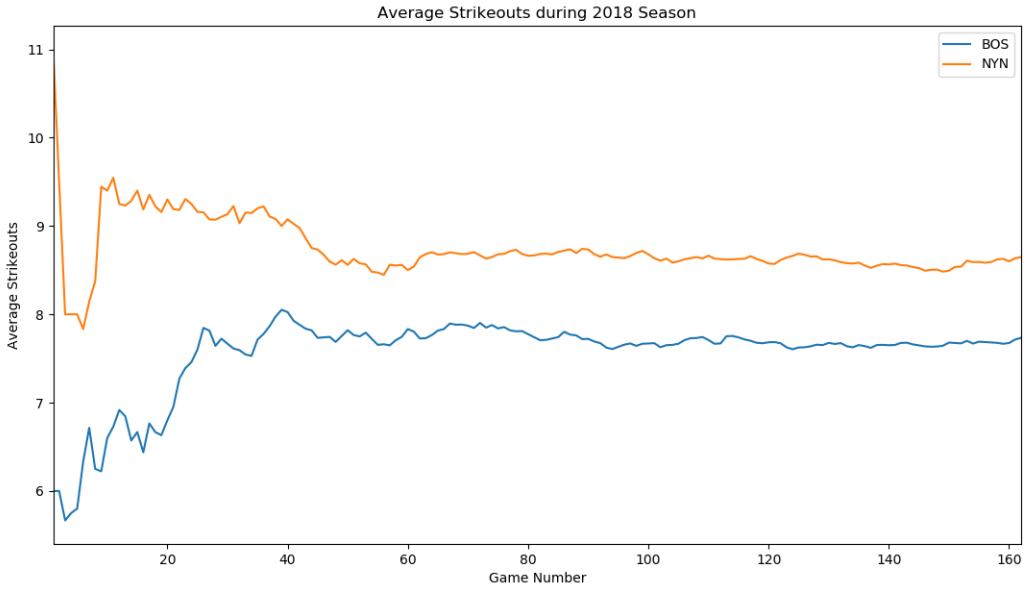


Figure 4: Difference in strikeouts between Red Sox (Blue) and New York Mets (Orange)

Again to further invesitgate the effect that the difference in offensive strikeouts has on win percentage, the average strikeout values were plotted. The same teams from before were used again to show the difference in average strikeouts between a good performing team and a bad performing team. As seen in Figure 5, the Mets really are not as good of hitters as the Red Sox because in 2018, on average, the Mets struck out more than the Red Sox. Using our previous finding that strikeouts are negatively correlated with a win, we can infer that the Mets are more likely to lose their games. Using the previous values from Baseball-Reference, we can see that the Mets lost more games in 2018 than the Red Sox. If these two teams were to matchup, we could assume that the Red Sox would win based on the differences in the statistics from both teams.

OPS vs Win

The last varaible that was looked into was OBP, which measures percentage of the time a batter can get on base. In Figure 6, the average OBP values for each team at the end of the 2018 was collected and plotted in a histogram. We can see that the plot is rather normal but may need some normalization if statistical analysis was needed. In the plot, one can see that the Res Sox's average OBP value is one of the largest average OBP values in the entire MLB in the 2018 season. Plotted for comparison is the Dodgers, who came up with a higher than average .323 OBP value. We can see that compared to everyone and the Dodgers, the Red Sox were extremely effective offensively that season. Their effectiveness can be seen in the amount of

games they won in the 2018 season, which was referenced a few paragraphs ago. Again, we can infer from this graph that because the Red Sox had the highest OBP value in the MLB, that they would go on to win the World Series for that year. After checking Baseball-Reference we see that the World Series Champion in 2018 was the Boston Red Sox, which confirms our assumption.

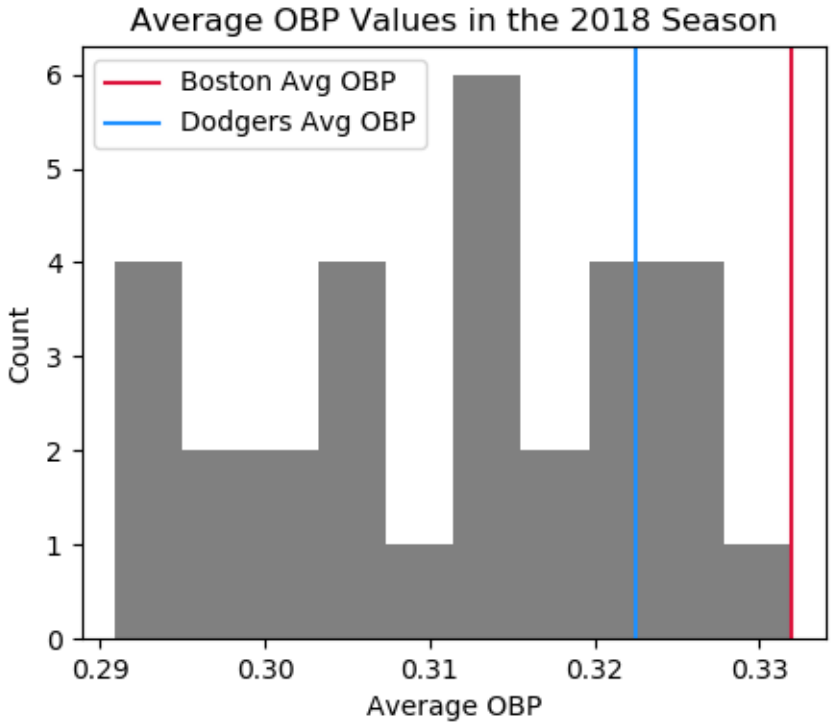


Figure 5: Histogram showing ditribution of average OBP values over the 2018 season

5 Modeling

For this project two models were used for two different goals. One model used was logistic regresiion for the classification task of predicting win/loss and the other model was linear regression which was tasked with predicting the score of games. The first that will be discussed is the logistic regression model for predicting the winner and loser of a game. This will also include comparing the training results of the unprocessed data and the processed data to show the difference that data pre-procssing can make.

Logistic Regression - Unprocessed Data

The combined unproccessed data was used in this step which included over 40,000 rows and about 75 columns, many of which were already removed. As seen and discussed before, the sheer amount of variables causes there to be no variables that correlate with a win. Obviously, this drastically reduced the logistic regression model performance in the training stage. The data was modified into an 80/20 train/test split which saved 80% of the dataset for training and hid away" 20% for testing. Using the unprocessed data, our logistic regression model got **50.4%** of the game predictions correct in the training phase. This is not too bad because this probability tells us that the system is basically picking games at random. So at least with the unprocessed data we can get a relatively good performance for not having done anything with the original data.

Logistic Regression - Processed Data

Once the data was thouroughly pre-processed it was ready to be fed into a new logistic regression model. The same procedure of training and testing split was used for the pre-processed data as well. However, before the pre-processed data was trained on, the feature importances for each variable was calculated. These feature importance rankings would help reduce the size of the dataset even more because they indicate unneccesary variables. For predicting a win it was found that Doubles were the least important value when predicting a win. Using this information, doubles were dropped from the dataset before training. Reducing the dataset in the pre-processing stage and with feature importances allow us to uncover more variables that are correlated with a win such as OBP and OPS. After all of these preliminary steps, the logistic regression model was trained and achieved a **98.77%** correct prediction rate on the training data. This is a **48.3%** performance increase from the unprocessed data to the processed data. This result shows the importance that data pre-processing has on machine learning and the performance gains one may expect to see when doing data pre-processing. However, the true test of the power of data pre-processing can be found when using the test set as this mimics the model's predictive abilities on data it has never seen before.

Linear Regression - Processed Data

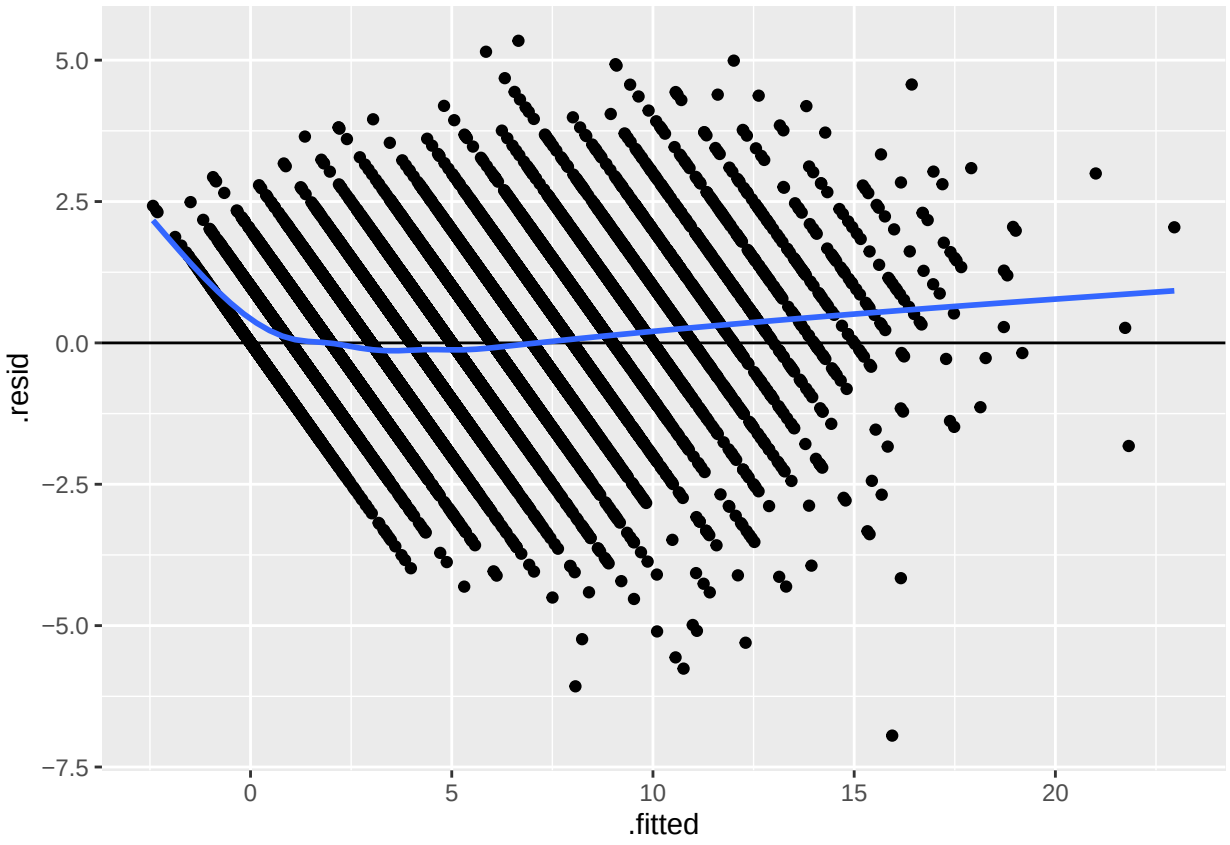
This model was tasked with predicting the scores for upcoming games much like the previous two models. However in this case, the prediction is a number and therefore needs a regression model like linear regression. Again this model used the same 80/20 train/test split just like the previous two models. Because this model was written R, many different diagnostic plots and statistics can be calculated and interpreted for the model. One of these is the p-values for each variable in the training dataset. In this model, it was found that a large majority of the original variables, like doubles, had slope estimates close to 0 with a very small p-value, meaning that the estimates are statistically significant. Most of the feature engineered variables, such as OBP or batting average, were found to have large slope estimates and very small p-values, signifying that they are statistically significant.

```
## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------ t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma

## Loading required package: mosaic

## Loading required package: ggformula

## Loading required package: ggstance

##
## Attaching package: 'ggstance'

## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh

##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")

## Loading required package: mosaicData

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following object is masked from 'package:boot':
##
##     logit
```

```
## The following object is masked from 'package:caret':
##
##     dotPlot

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     stat

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median,
##     prop.test, quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum

##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa

##
## Attaching package: 'cutpointr'

## The following objects are masked from 'package:caret':
##
##     precision, recall, sensitivity, specificity

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine

##
## Attaching package: 'olsrr'

## The following object is masked from 'package:datasets':
##
##     rivers
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



The next diagnostic plot that was created can be found in Figure 7, which is a residuals vs fitted plot. This plot uses the predictions made by the trained model and displays them with their respective residual values. In this case the residuals are the actual values minus the predicted values, or in mathematical terms $y - \hat{y}$. What we want from this graph is for it to be equally distributed across 0 on the x axis, or in other terms a slope of 0. If this is the case then the data is linear and can be well approximated by linear regression. In the figure we can see that the data has an oblong shape but looks generally well distributed across the x-axis. However there is some non-linearity around both ends which could signify that we need to omit some outliers from the dataset. However, the graph looks linear enough for us to say that our baseball data is well approximated by a linear model and thus can move onto more training. After training the model the root mean squared error, $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y-\hat{y})^2}$ was calculated for all the predictions made during training. We caluclated the RMSE value to be **1.045** which signifies that on average, the predictions made by our linear regression model are 1 point off of the actual score. In the subsequent sections, we will employ cross-validation to estimate a test error and check whether or not the model is overfitting the data.

6 Model Evaluation

To evaluate the models, 10-fold cross validation was used to assess the predictive abilities of both the regression model and the classification model. Doing 10-fold cross validation will allow us to estimate the test error without ever using the test data. This is done using the training data and multiple repeats to come up with a population error value which will become our estimated test error. For the regression model, 10-fold cross validation calculated an estimated test error of **1.05** which was slightly higher than the training error which is expected. For the classification model using processed data, the 10-fold cross validation calculated the test accuracy to be **98.76%** which, again, is slightly lower than the training accuracy. Once the estimate was calculated, we attempted to predict the winners and scores of games in the heldout test data and 15 actual games in the 2019 season. If the model performs like it did in the training and in the cross validation, the results for the test data should be promising.

7 Results

Logistic Regression - Unprocessed Data

As seen before, the logistic regression model trained on unprocessed data produced poor results on the training data. When test data was used, the results were even worse than the training data. In this case the model obtained a **48%** accuracy rate on the unprocessed data. This was expected due to the poor performance on the training data. Figure 8 depicts the confusion matrix for the model using unprocessed data. The confusion matrix lets us know what kind of mistakes the model makes when it comes to the test data. Using the confusion matrix we can see that the model predicts more correct wins then losses but still has a very low accuracy rate.

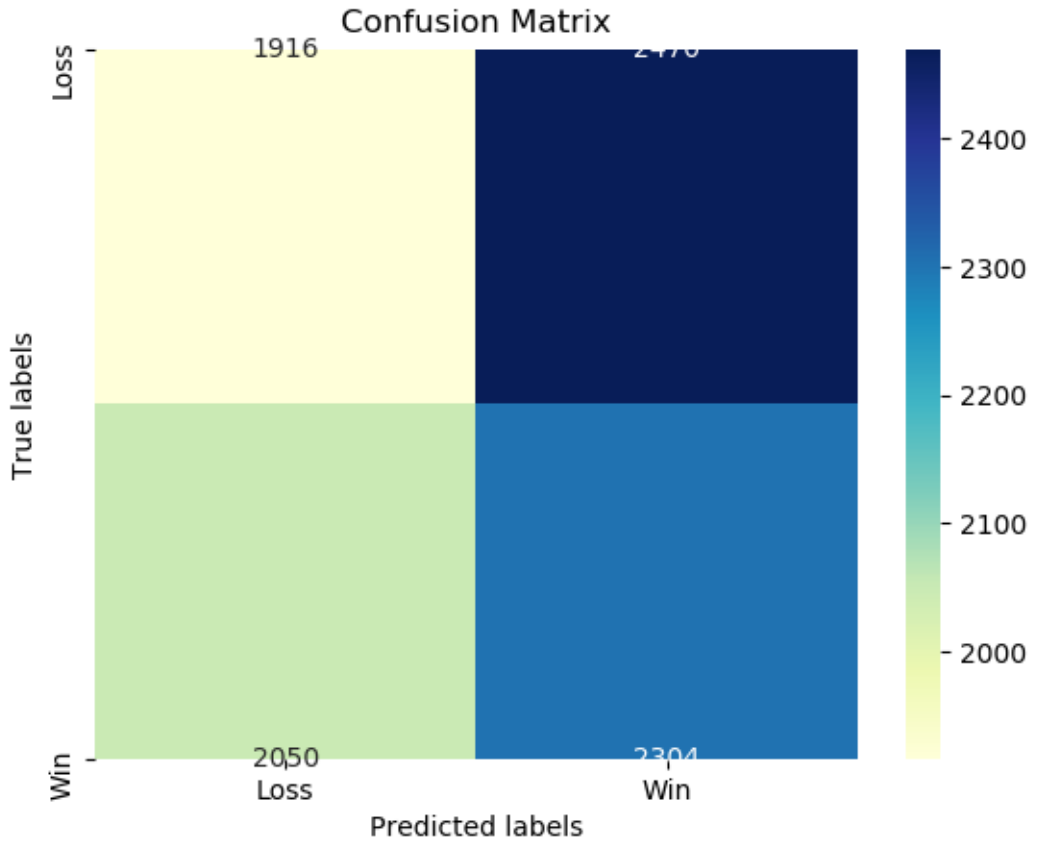


Figure 6: Confusion matrix for the predictions made by the logistic regression model trained on unprocessed data

Logistic Regression - Processed Data

After discovering the poor performance of the unprocessed data, the same procedure was done for the processed data wtih additional metrics. In this case the model was trained on the processed data and then the test processed data was fed in for predictions. For this model, the test accuracy that was calculated was **98.65%**. This value is slightly lower than both the training accuracy and the cross-validated estimate for the accuracy. Training on processed data increased our test error by about 50% which is an incredible boost in performance. Checking out the confusion matrix in Figure 9 shows us that the model predicts the same amount of correct losses and wins and rarely makes incorrect predictions. When the model was applied to the Opening Day 2019 games, the model achieved a **73%** accuracy on these real games. The sharp decrease in accuracy is explained by the use of 10-game averages to make predictions. Because we did not have the data for the games like we did in the training, we had to synthesize data to predict off of. This obviously caused a sharp decrease in performance but may be able to be salvaged by using different models in later iterations of the project.

```
##              precision    recall  f1-score   support
##
##           0       0.99      0.98      0.99      4013
##           1       0.98      0.99      0.99      4085
##
```

```
##     accuracy                           0.99      8098
##    macro avg       0.99      0.99      0.99      8098
## weighted avg       0.99      0.99      0.99      8098
##
## [[3937   76]
##  [  33 4052]]
```

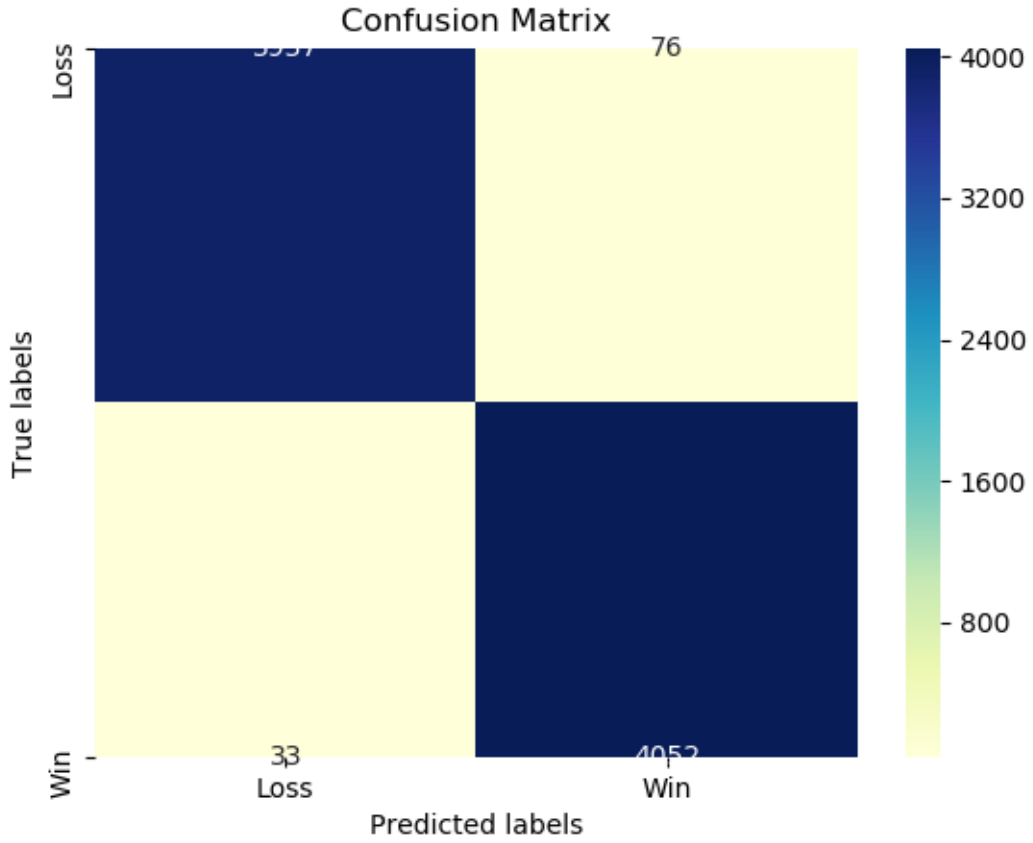


Figure 7: Confusion matrix for the predictions made by the logistic regression model trained on processed data

To further assess the model an ROC curve was constructed and is found in Figure 10. This curve found in the graph is used to assess the threshold value for logistic regression. If the threshold value is optimal then the curve will follow the two dotted black lines found in the figure, almost a 90 degree angle. For our model, the ROC curve follows these dotted lines almost exactly showing that our threshold value is optimal for the given problem. Plotted for reference (red) is the ROC curve for a logistic regression model that just does random guesses (ie a 50% accuracy rate). Our model heavily outperforms a random guess logistic regression model and this confirms that the current threshold value of 0.5 is optimal.

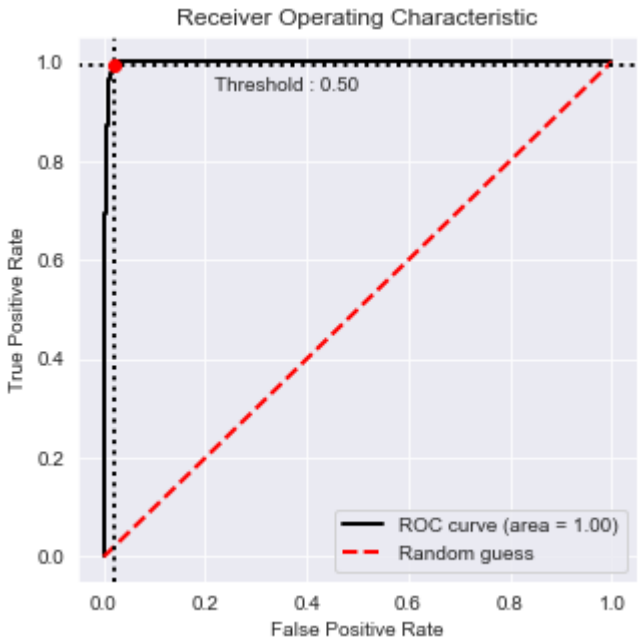


Figure 8: ROC Curve generated for the logistic regression model trained on processed data

Linear Regression - Processed Data

8 Conclusion