BETTING BASEBALL

Capstone Final Report

Atticus Soane

TABLE OF CONTENTS:

Can You Beat Vegas From Your Laptop?

Figure 1: headerimage

There's an old adage in gambling: The House always wins. And while many grandmothers may claim a a slot machine strategy that could bankrupt any casino, this isn't changing anytime soon. They make the rules and they make them so that they will always win. But with the legalization of sports gambling in the United States (at the Federal level at least) and the LEGAL emergence of sports gambling as a multi-billion dollar industry, can an average person leverage machine learning to gain an edge on Vegas oddsmakers? I, an average person, chose to investigate. For the purpose of this investigation, I focused on Major League Baseball - a notoriously difficult league to bet successfully. There were two major guidelines for this investigation.

1. Does the model yield profit? As will be discussed later, it is possible to predict outcomes accurately and still lose money. Any useful model must generate **PROFIT**.
2. Can the model be built with data that is publicly accessible? There is information that can be purchased that would be considered "Inside Vegas" information: ticket counts, money percentages, etc. That is

outside the scope of this investigation. I will only use data that is public domain.

Now before we dive in, a quick sports gambling primer for those unacquainted with the sports gambling space. There are far too many different types of bets to try to explain here: if you want to go find some value in **Compact Win or Tie Only** bets more power to you. This project will focus on one of the three most common baseball bets: Betting the **Moneyline**.

- **Moneyline**:

  This is the most basic of all sports bets: the bettor is simply betting on which team will win the game. The score is of no consequence as long as the bettor correctly picks the winning team. The payout of this bet is simply determined by the line offered for the winning team. This line is nothing more than a transformation of an applied win probability:

$$Line = \frac{1}{P_w}$$

  where P_w is the implied win probability for either team. Typically, the lines displayed are called American style: they indicate the return on a 100 dollar wager. A negative line value indicates the betting favorite: the team expected to win. The return on such a bet would be calculated as follows:

$$\frac{100}{-1 * line_{fav}} * BettingUnit$$

  For example, if the line offered for the winning team was -150, the payout for a 100 dollar bet would be calculated as follows:

$$\frac{100}{-1 * -150} * 100 = 66.67$$

  As you can see, this is not a 1 to 1 payout. The difference between the actual payout and what a 1 to 1 payout would be is called the **VIG** and is often colloquially referred to as "juice".

  Now, if the line value is greater than zero, this indicates the betting underdog. This is a much simpler calculation:

$$\frac{line_{underdog}}{100} * BettingUnit$$

  So, if the line was $+150$, the payout for a 100 dollar wager would be:

$$\frac{150}{100} * 100 = 150$$

DATA COLLECTION

There is a virtually limitless amount of baseball data available going back to the founding of the leag

1. __NOAA Global Historical Climatology Network__-

   Using the GHCN FTP, daily weather observations were collected for a myriad of weather stations thr

2. __Retrosheet__-

   The vast majority of the baseball data collected was collected from the vast Retrosheet database. T

3. __FiveThirtyEight__-

FiveThirtyEight maintains an Elo dataset which contains a number of team and pitcher ratings genera

4. __Baseball-Reference__-
   Baseball-Reference is another treasure trove of information. I collected stadium names and location

5. __SportsBookReview__

   This data was not included in initial models (I was not able to locate historical gambling odds goi

6. __FanGraphs__

   FanGraphs maintains extensive statistics and records. Many of the statistics used have multipliers

FEATURE SET / FEATURE ENGINEERING / FEATURE IMPORTANCE

TARGET FEATURE

As with any machine learning problem, the first question to answer is to determine what is being predicted. There are many different ways to frame this problem, but for the purpose of this investigation I elected to create a target feature HOME WIN - a simple binary feature with level 1, the home team won, or 0, the home team lost. The reason for using HOME WIN as opposed to shuffling the data randomly and assigning a binary target feature was two-fold. During data exploration there were two interesting patterns:

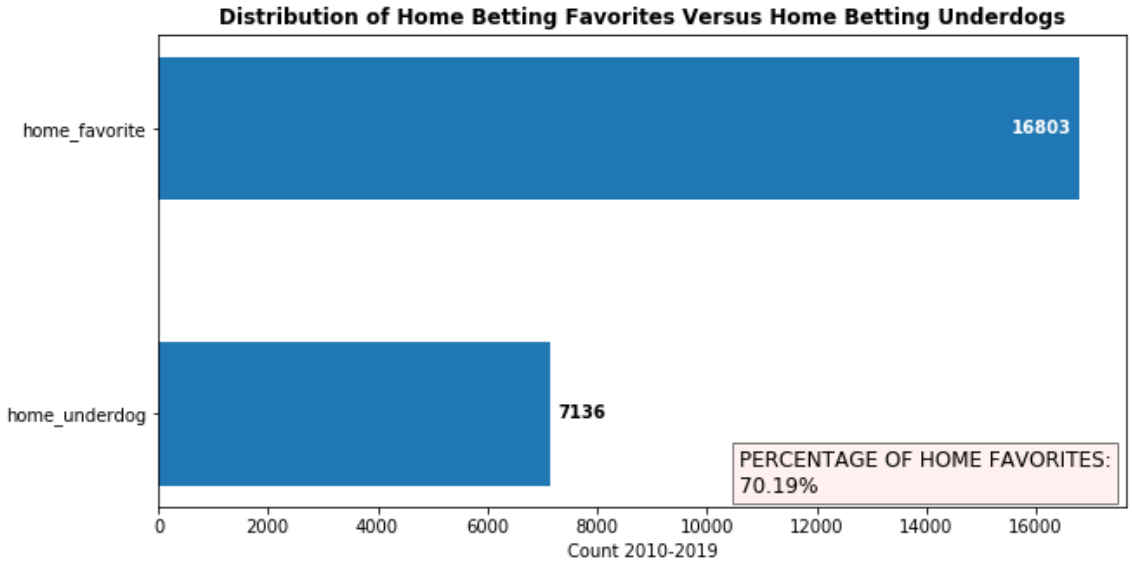

Figure 2: HomeFavDist

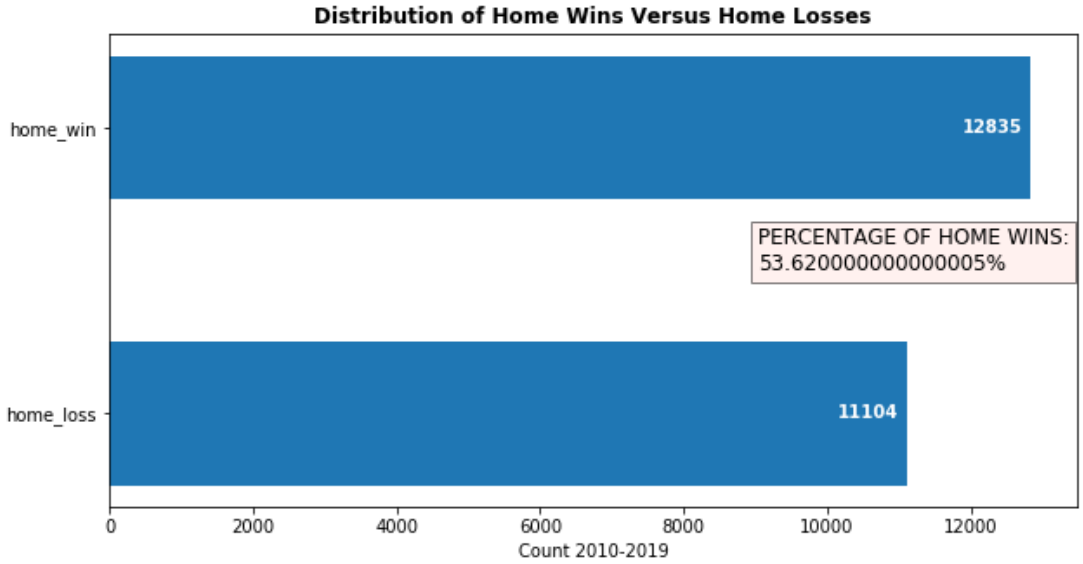

Figure 3: HomeFavDist

The home team is set as the betting favorite over 70% games, but home team only wins 53% of these games. That means that road underdogs are winning 47% of their games, which is actually a higher percentage than home underdogs; home underdogs only win games at a little over a 43% clip:

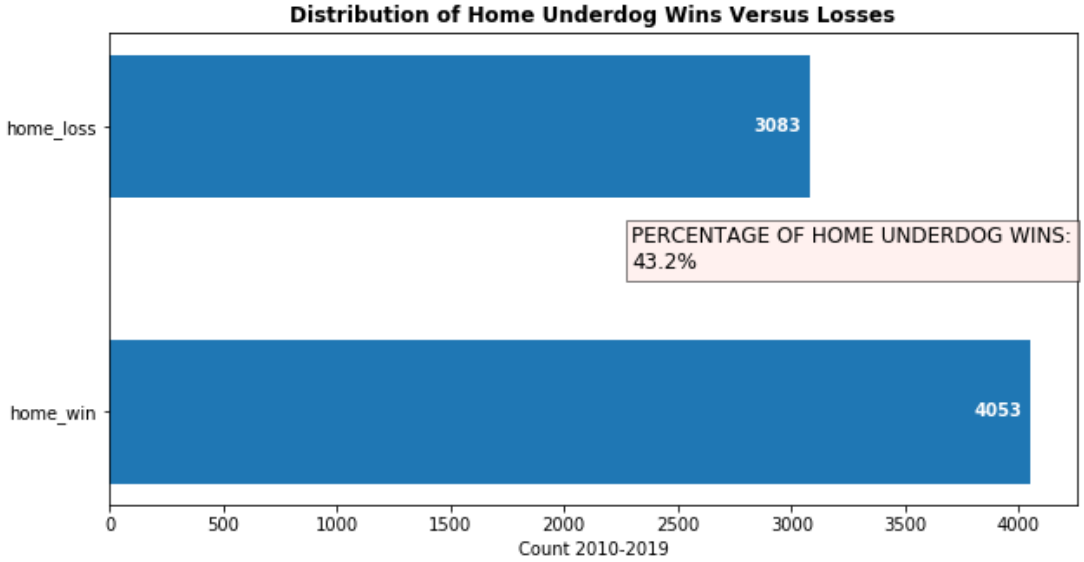

Figure 4: HomeDogsDist

This is both counter-intuitive and presents an area to potentially exploit. If this discrepancy in home team favorites and home team wins can be found, there is potential to generate profit in that window.

FEATURE SET

The feature set considered is too large to list here, but the features fall in to several categories:

- TEAM BASED STATISTICS - Team offensive and relief pitching statistics.
  - The offensive statistics analyzed were: OPS, wOBA, wRAA, wRC, runs per game and hits per game. The relief pitching statistics chosen to anlayze were: K/9, K/BB, WHIP, ERA, FIP, wOBA against, wRAA against, and WRC against. It would be another paper entirely to explain all of these but please see https://library.fangraphs.com/offense/ for detailed explanation of the calculation and meaning of these statistics.
- STARTING PITCHER STATISTICS - Statistics for the starting pitcher of each team.
  - Statistics considered were K/9, K/BB, WHIP, ERA, FIP, wOBA against, wRAA against, and wRC against. Please see https://library.fangraphs.com/pitching/ for reference.
- FIVETHIRTYEIGHT RATINGS - Ratings used in FiveThirtyEight modeling. Please see: https://fivethirtyeight.com/features/how-our-mlb-predictions-work/, a detailed explanation of the FiveThirtyEight modeling technique.
- GAMBLING STATISTICS - Consensus betting lines that incorporate line movement. Monitoring the change between the opening betting line and the closing betting line can give a rough estimate of how bettors predicted the game.
- EXTERNAL FEATURES - Attendance, weather, distance traveled, factors accounting for different ballparks, etc. These are values independent of the teams on the field.
- MOMENTUM - While it is impossible to quantify momentum, these features made an effort to do so. These features reflect a team's recent performance and trends.

FEATURE ENGINEERING

As briefly mentioned in the Data Collection section, all team and pitcher statistics had to be created. While there are many records of season-end and game-end statistics, for the purpose of predictive modeling I needed each teams' statistics and each pitchers' statistics on every day of the season, as well as season trends for each team.

FEATURE SELECTION

The models discussed in the modeling section do not use the same feature set and thus there is not one single feature set. During experimentation, different feature sets were selected using recursive feature importance-based selection. Using a suite of machine learning algorithms, a feature set is evaluated using K-Fold cross validation and a specified metric (Area Under ROC curve in this case, but it could be any desired metric), features are removed that are not contributing to the model (either features with low feature importance for tree based algorithms or features with small coefficients for linear algorithms), and then the feature set is re-evaluated until the feature frame can no longer be reduced. Different algorithms emphasized different features, but the most frequently appearing features are:

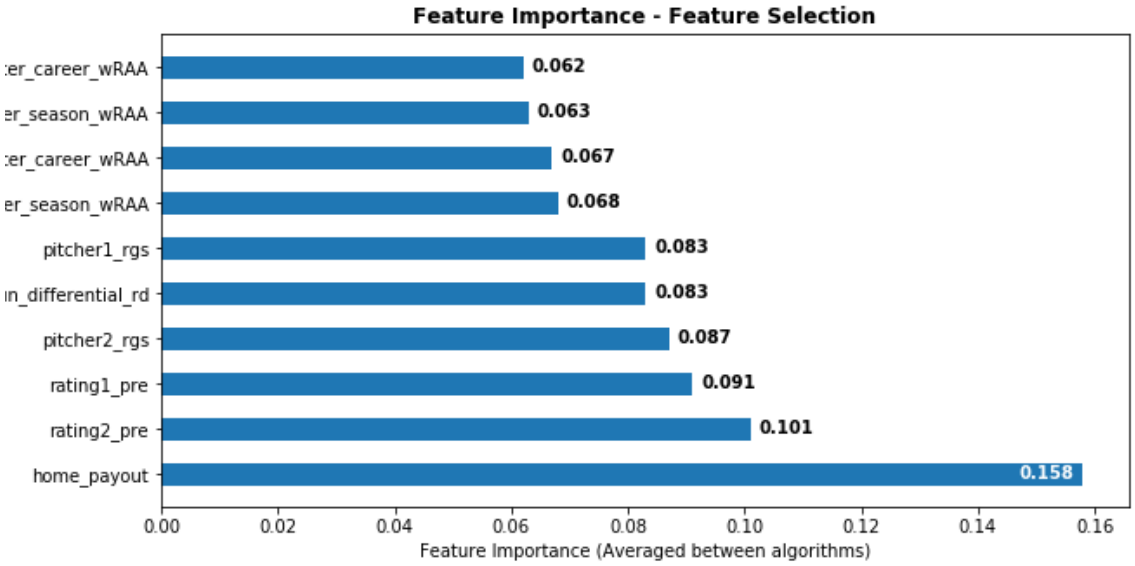

Figure 5: FeatureImportance

A major issue during this process was collinearity and multicollinearity.

Many of the team and pitcher statistics are highly correlated, but removing them resulted in information loss, as the features most important to the model were also the highly collinear and multicollinear features. Multicollinearity was calculated using the Variance Inflation Factor:

$$VIF_{\hat{\alpha}_i} = \frac{1}{1 - R_i^2}$$

where R_i^2 is the coeffiecient of determination for each feature α_i with respect to all other features - the proportion of variance in each feature that can be explained by all other features. A thorough explanation can of VIF can be found here: https://en.wikipedia.org/wiki/Variance_inflation_factor

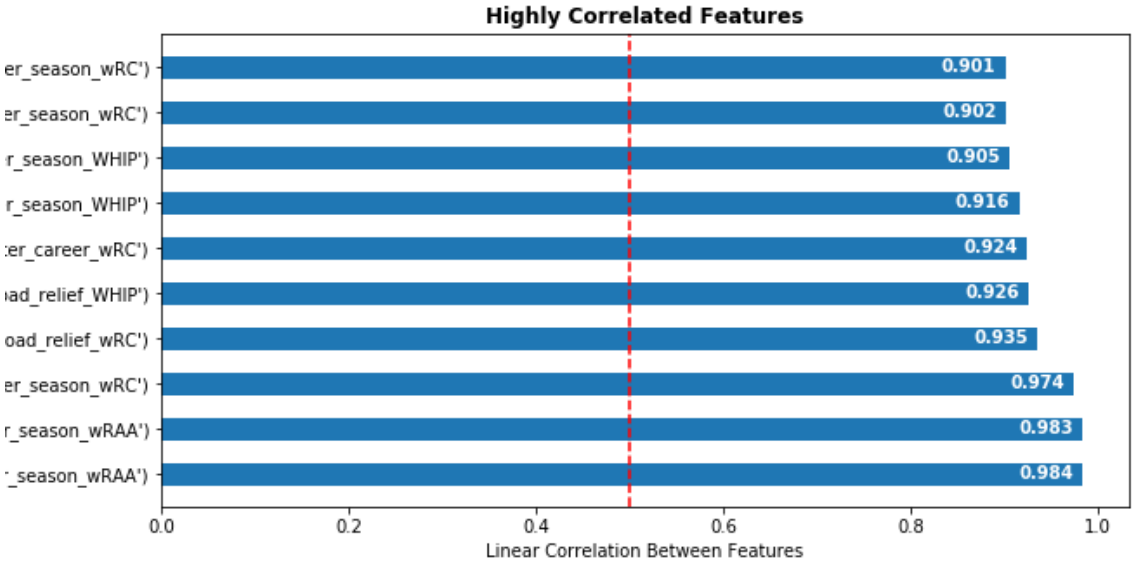

Figure 6: Collinearity

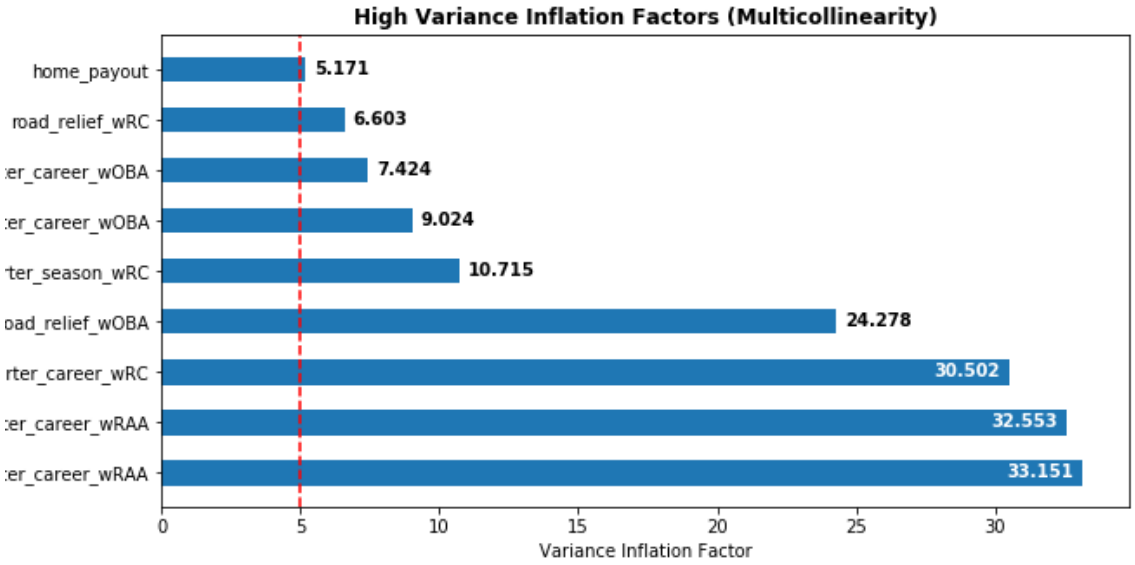

Figure 7: Multicollinearity

To overcome this, the final models are primarily centered one one statistics: wRC+. wRC+ is a 100-based statistic and thus it allows for relative comparison on a single scale as relative comparison between eras. Using the 2019 season, it can be seen that statistic is an effective rating tool of performance that provides a smooth distribution:

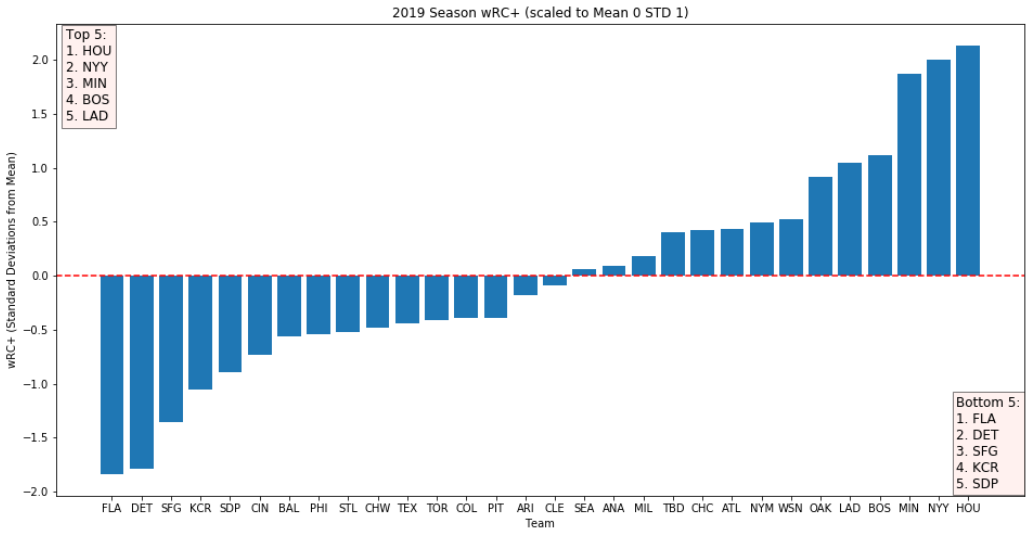

Figure 8: 2019wRCDist

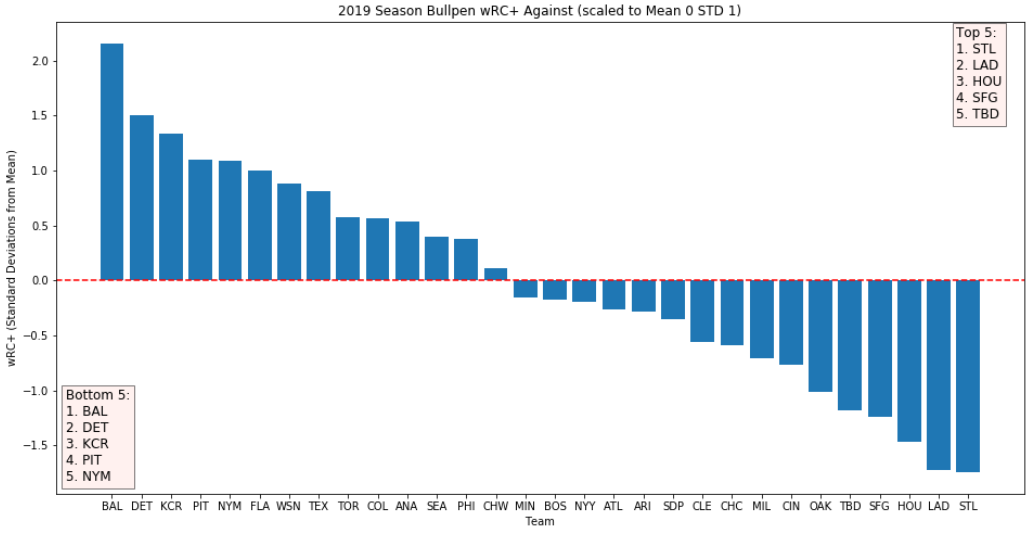

Figure 9: 2019wRCBullpenDist

Modeling

KPI / Evaluation Metrics

Looking from the outside in, it would seem that generating a gambling model based on team wins and team losses is a simple binary classification model. Binary classification has many different metrics to evaluate model performance: Accuracy, precision, F1-score, ROC Area Under the Curve, etc. However, none of these are sufficient in evaluating a gambling model. It is more than possible to achieve a high accuracy (relatively, accuracy in predicting baseball outcomes is never that high) and still produce a betting scheme that loses money. The FiveThirtyEight predictions demonstrate this point perfectly. FiveThirtyEight, as mentioned previously, maintains the MLB Elo model which is an extremely popular and successful baseball model and is considered somewhat of a gold standard in predictive power. For the 2019 baseball season, the FiveThirtyEight model accurately forecasts 59.82% of baseball games. While experimenting with models that strictly predict game outcome, several models were able to predict with very similar results, accurately forecasting 59.99% of games. But look at the profit generated by the FiveThirtyEight scheme:

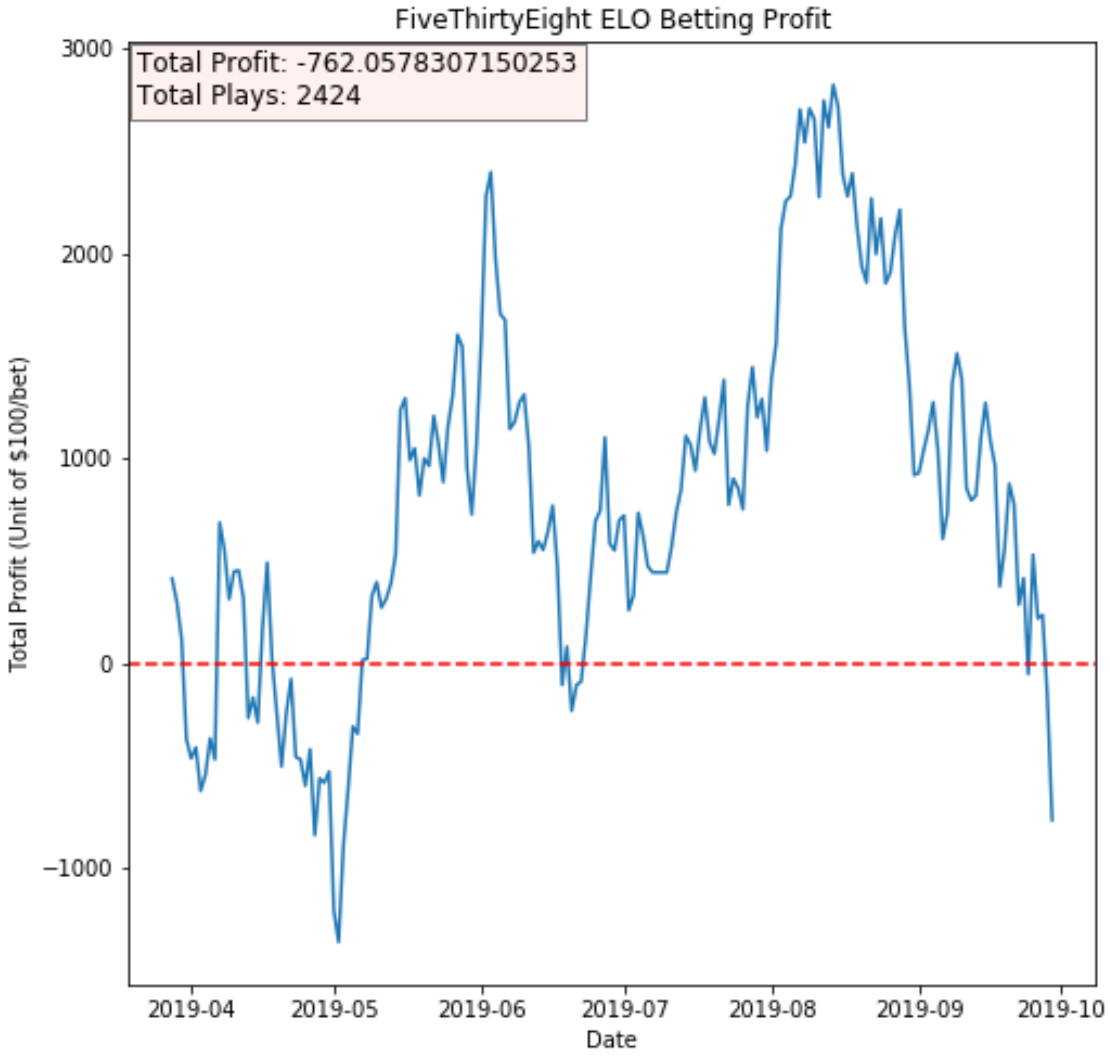

Figure 10: EloBase

Even though the games were predicted with almost 60% accuracy, the betting scheme still loses money on the season. You'll also notice that the results are not a tremendous improvement upon just flipping a coin to decide each game.

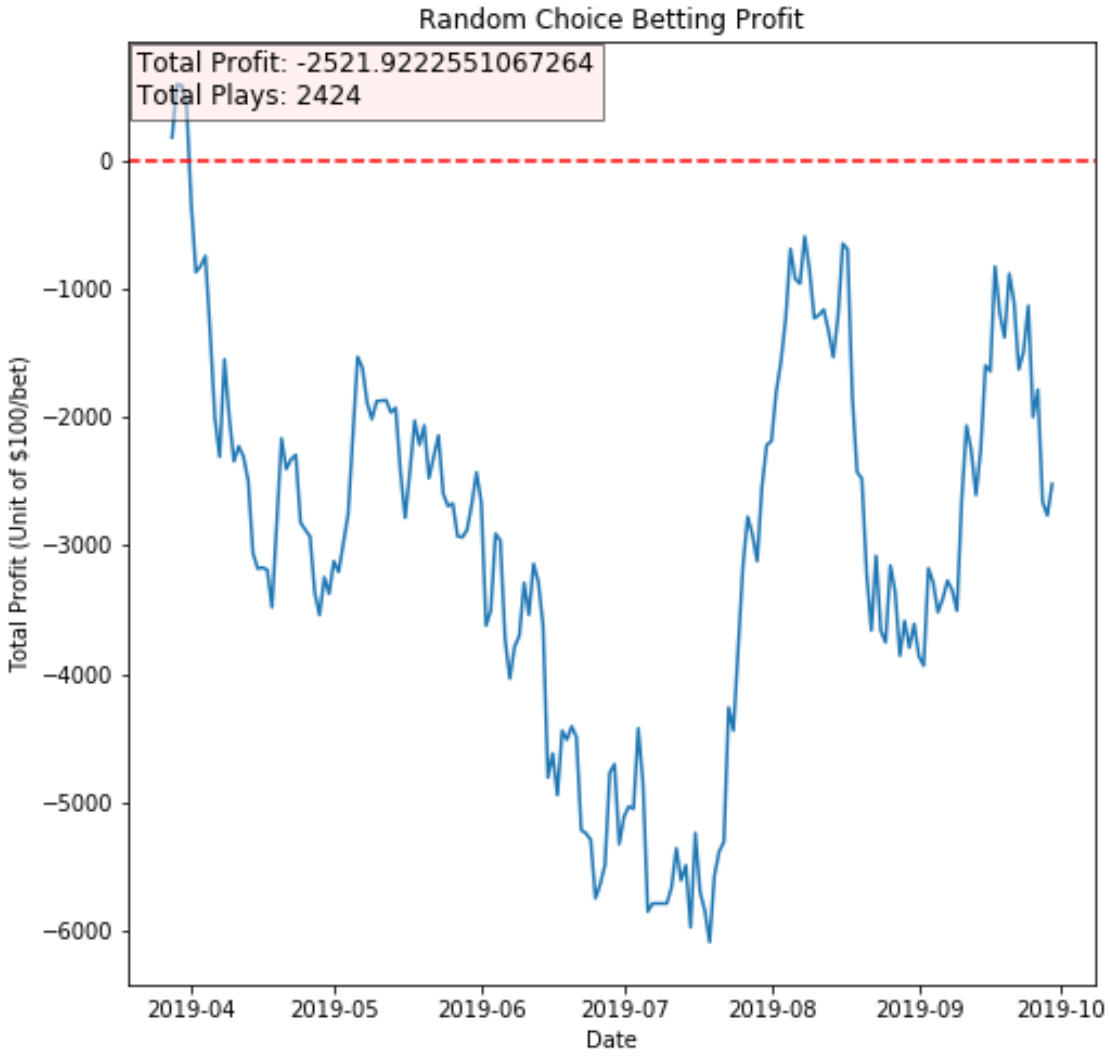

Figure 11: RandomChoice

The reason being is the aforementioned VIG: Betting favorites, who win more frequently, yields lower payouts. The only metric that matters in evaluating betting schemes is bettor profit, and therefore this will be the only metric considered in evaluating models.

Methodology

There are many different ways to frame and approach this problem. The problem can be framed as a Regression problem, a Binary Classification problem, or a Multi-Class classification problem. While the strengths and shortcomings of these different methods are interesting in their own right, in the interest of time I will only discuss two methods that showed promising results.

A Note on Data

The two models outlined below are trained on different feature sets. I will include the features used in each model and the data preparation techniques within each model. However, the commonality of both models is that they were trained on data from past seasons and used to predict the 2019 season. The point is to use past seasons to build a model to predict the next season, so randomly splitting into training and validation sets did not make sense in this case.

Deep Learning - Binary Classification with Gaussian Rank Scaler

Notebook reference: GAUSSRANK_FEEDFORWARD_NN.ipynb

The technique that showed the most promise was a Fully Connected Feed Forward, Back Propagation Deep Neural Network. This is the only model that was trained on the entire feature set. Each continuous feature was prepared with a technique called GaussRank scaling. The algorithm is as follows:

- Each unique value in given feature α_i is ranked and forced into a distribution between $[-1, 1]$.
- Sufficiently small ϵ is chosen and applied to $min(\alpha_i)$ and $max(\alpha_i)$ such that the domain endpoints are not inclusive: $(-1, 1)$
- The inverse error function is applied to each value. The Gaussian error function is defined as:

$$erf(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt$$

  This represents the probability that for a random variable Z with $\mu = 0$ and $\sigma = \frac{1}{2}$, $erf(x)$ is the probability that Z falls in the range $[-x, x]$. The inverse of this function maps the ranked values to a Gaussian distribution.

The GaussRank scaler for several reasons. Rank transformations are robust to outliers and there were large outliers in the dataset, particulary in the beginning of seasons. However, they were not removed because the observations were accurate. There also was huge differences in variance of features, and thus normalization is beneficial.

One drawback of the GaussRank scaler: the implementation of the GaussRankScaler that was used https://github.com/aldente0630/gauss_rank_scaler does not extrapolate. Intuitively, this makes sense as a value outside the feature's distribution would be outside of the ranked values. Thus, when the transformer object was fit to the training dataset and used to transform the test dataset, a handful of observations were dropped.

The model was trained using Keras, a high level Python API to train Neural Networks on top of the powerful Tensorflow library.

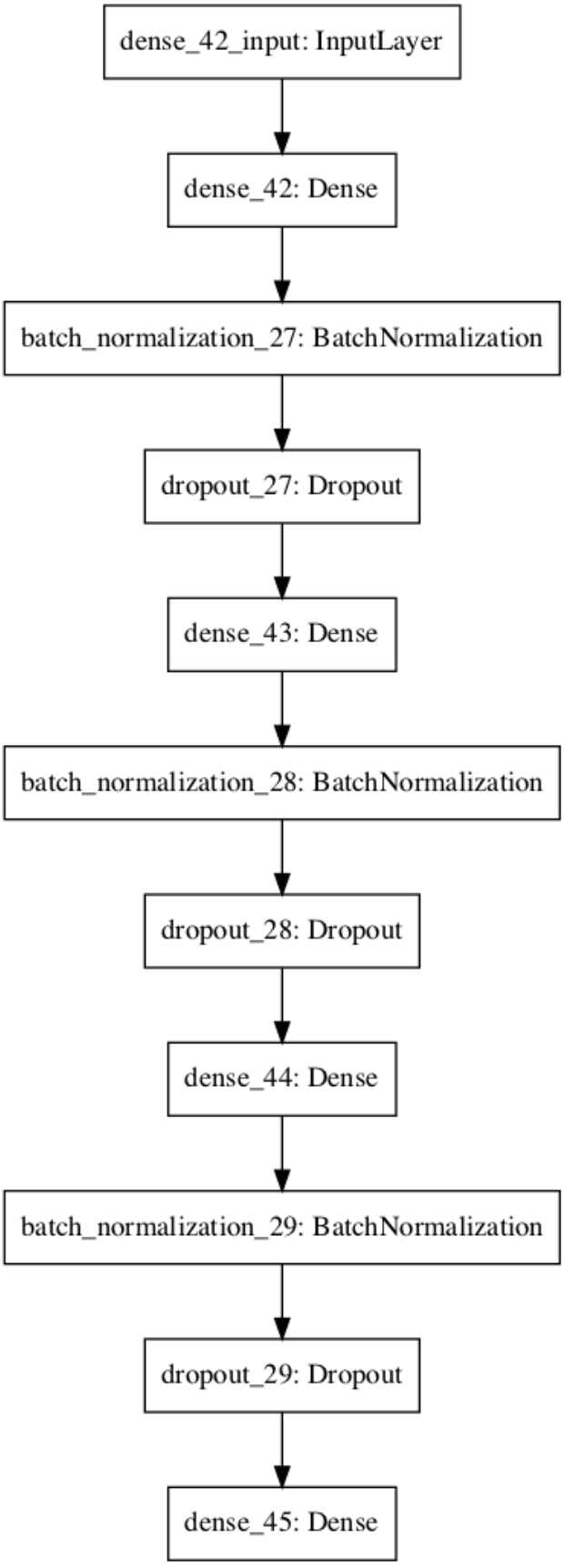

Figure 12: nn_arc1

The architecture for the model, seen above was as follows:

- Densely Connected Input Layer - ReLU activations
- Batch Normalization Layer - Scale activations to Mean 0 and Standard Deviation 1.
- Dropout Layer - Reset fraction of activations to 0.
- Dense Hidden Layer - ReLU activations
- Batch Normalization Layer
- Dropout Layer
- Dense Hidden Layer- ReLu activations
- Batch Normalization Layer
- Dropout Layer
- Dense Output Layer- Sigmoid activation

Each densely connected hidden layer contained 250 neurons. A Dropout layer fraction of 0.50 was used in each dropout layer to avoid overfitting. Weights were initialized using the Glorot Normal Distribution:

$$Weight_i \in \left[-\frac{\sqrt{6}}{Units_{input} + Units_{output}}, \frac{\sqrt{6}}{Units_{input} + Units_{output}} \right]$$

The model was compiled and trained with the following hyperparameters:

- Objective function: Binary crossentropy (log loss):

$$L\Big(y, \hat{y}\Big) = \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} log(\hat{p}^{(i)}) - (1 - y^{(i)}) log(1 - \hat{p}^{(i)})$$

- Optimizer: Adam - a stochastic optimizer, please see https://arxiv.org/abs/1412.6980v8 for a detailed explanation.
- beta_1 = 0.9 (optimizer param)
- beta_2 = 0.999 (optimizer param)
- Learning rate: 0.001 (optimizer param)
- Batch size: 512
- Epochs: 100

The resulting scheme was a vast improvement over the FiveThirtyEight model:

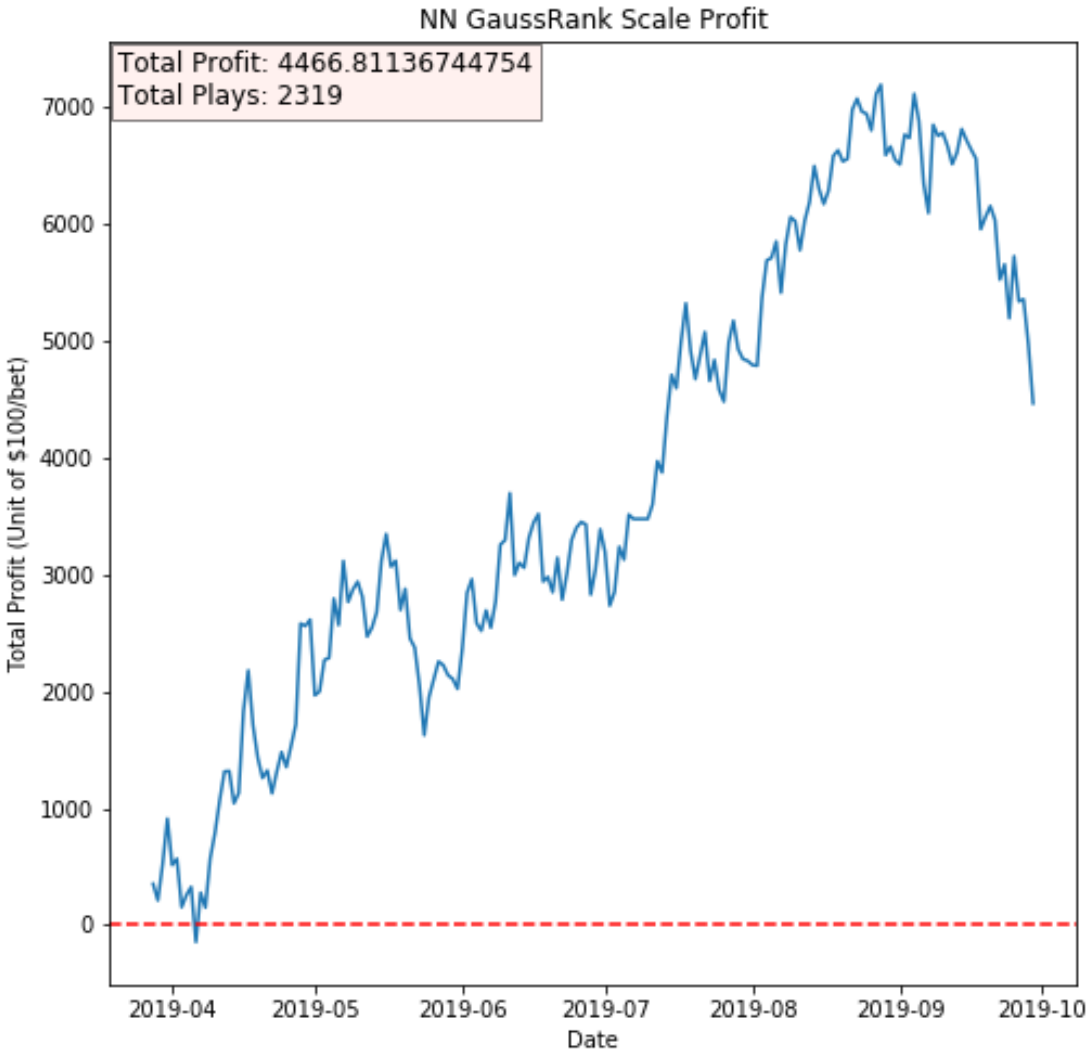

Figure 13: GaussRankFullPlot

The model as evaluated on the entire season as that was how the FiveThirtyEight model was evaluated, but if the model predictions are used to form an arbitrage strategy, there are other promising results.

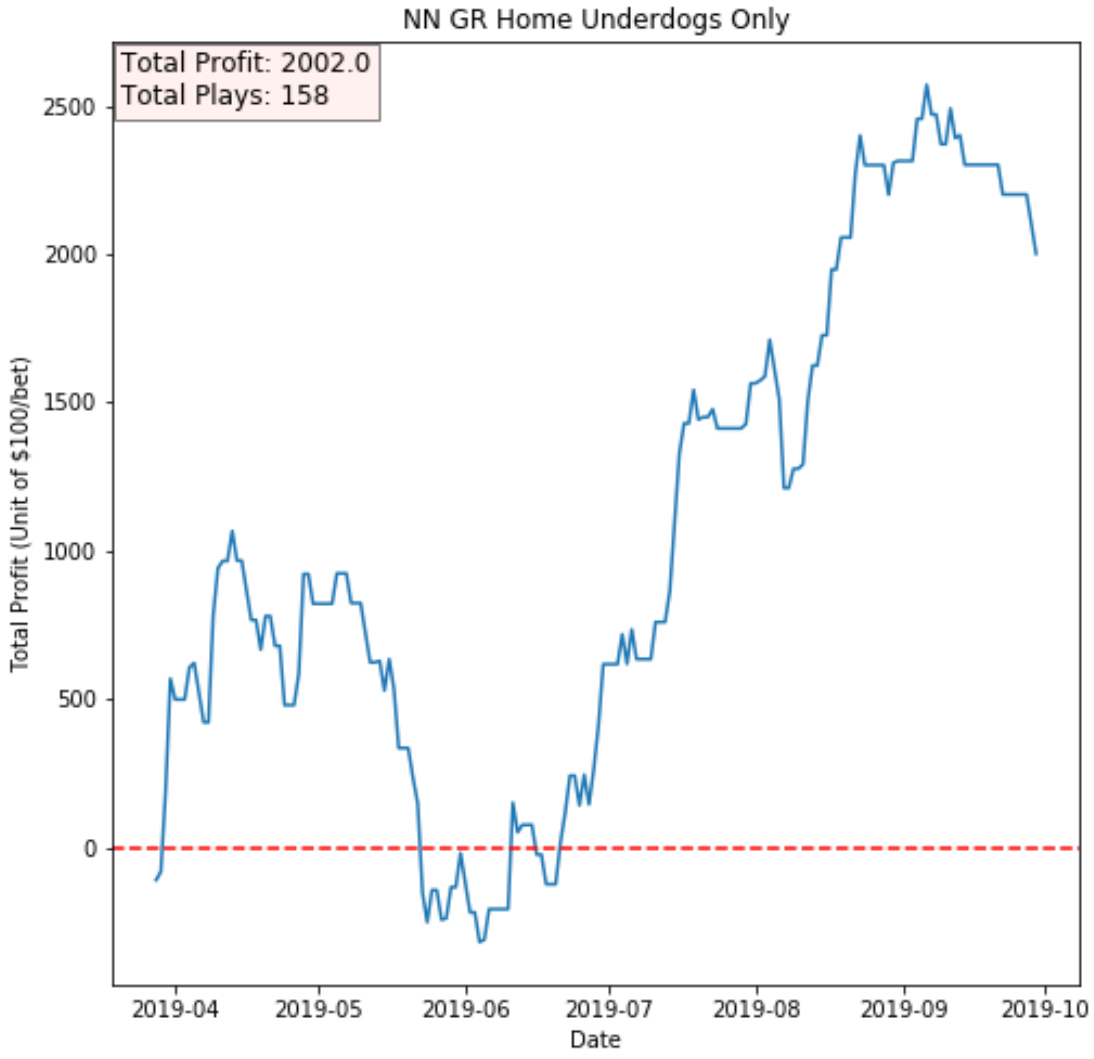

Figure 14: GaussRankHomeDogs

By betting on only home underdogs, the model is able to generate 44.8% of the profit with only 5% of the plays.

Keep in mind, a betting line is nothing more than an implied win probability. A betting line of -150 can be represented as follows:

$$\frac{1}{1+\frac{100}{150}} = 0.6$$

Thus, by applying this transformation to the probabilities returned by the model, it is possible to compare the lines the model generates to the actual lines and determine discrepancies.

Consider a simple arbitrage strategy:

1. If the model predicted home payout is lower than the actual home payout (the model thinks the home team should be a larger favorite), bet the home team.
2. If the model predicted home payout is higher than the actual home payout (the model thinks the home team should be a larger underdog), bet the road team

the results are better still:

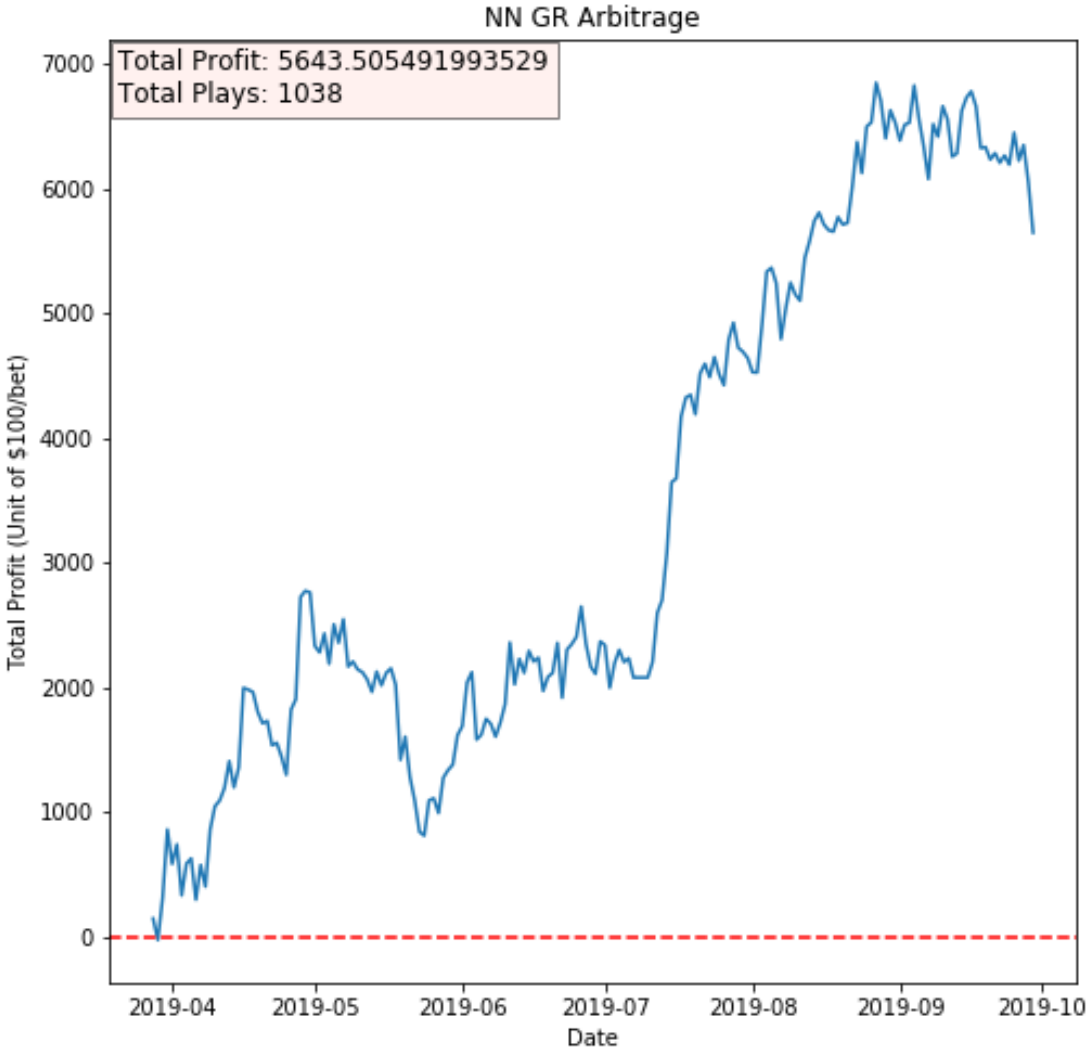

Figure 15: GaussRankArbitrage

Deep Learning - Binary Classification with Custom Loss Function

Standard loss functions used in Binary Classification, such as binary cross entropy used above, are not optimized for the task at hand. One way to work around this is to write a loss function that teaches a model how to identify good bets versus bad bets instead of wins versus losses. This loss function looks like this:

$$L\left(y, \hat{y}\right) = \sum_{i}^{n} \left(P_1^{(i)} * y - 1\right) * ReLU\left(P_1^{(i)} * \hat{y} - 1\right) + \left(P_2^{(i)} * (y-1) - 1\right) * ReLU\left(P_2^{(i)} * (1-\hat{y}) - 1\right)$$

where

- $P_1^{(i)}, P_2^{(i)}$ are the payouts of a correct bet placed on the home team and road team respectively.

The *ReLU* function (Rectified Linear Unit) simulates a betting strategy as it only "places" a bet when the value is greater than 0 and whereas other loss functions punish inaccurate outcome prediction, this loss function punishes bets with no value. An example to illustrate this: Placing a 100 dollar bet on a -340 betting favorite has a potential payout of 29 dollars if the betting favorite wins, while the potential is still 100. There is limited value to winning such a bet with the same amount of risk as betting on a +140 underdog, which would payout 140 dollars. Now, obviously, the -340 favorite will win more often than the +140 underdog, but this loss function searches for the bets with value.

The feature set used in this model was a fraction of full feature set: * Home Team and Road Team wRC+ * Home Team and Road Team Bullpen wRC+ against * Home Starter and Road Starter career wRC+ against * Home and Road Starter season ERA (Earned Run Average of Starting Pitcher, season) * Home record at home (Number of games above or below .500, 0-based) * Average Margin of Victory or Loss for Home Team and Road Team * Home opening moneyline (Betting moneyline for home team) * Road team record on road (number of games above or below .500, 0-based) * Current Streak for Home team and Road Team (Either record in last ten games, number of games above or below .500, or if they have won more than ten games in a row, number of consecutive wins)

The features were scaled to $\mu = 0$ and $\sigma = 1$.

The model was trained also trained using Keras. Keras has functionality for designing and implementing a custom loss function, which was critical for this model design.

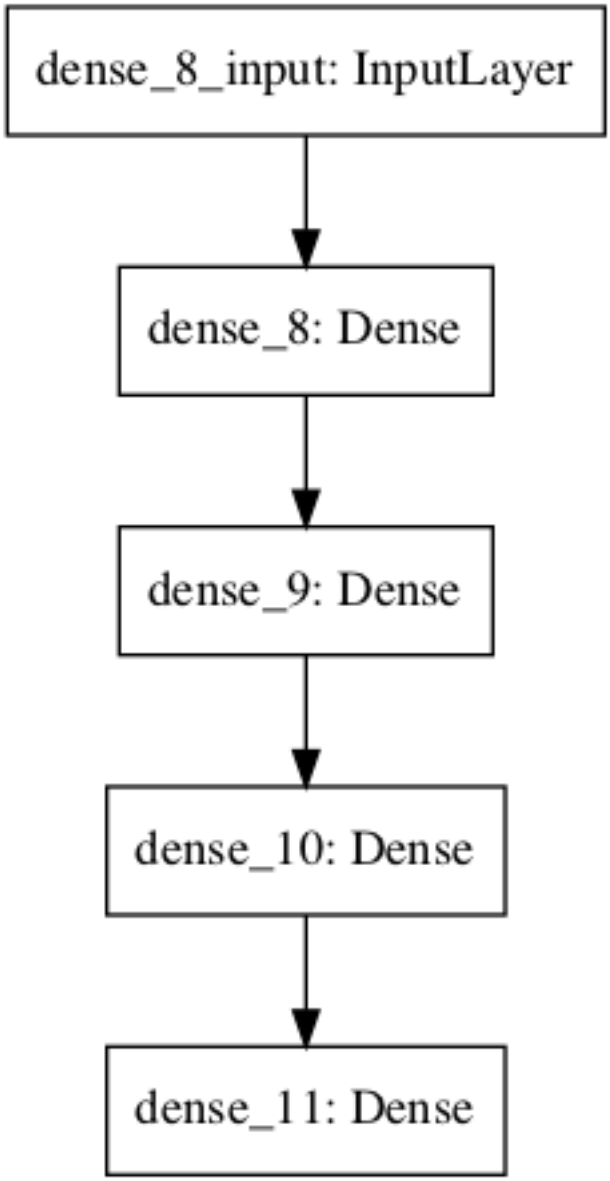

Figure 16: ModelArc2

- Dense Input Layer - ReLU activations
- Batch Normalization Layer - Scale activations to Mean 0 and Standard Deviation 1.
- Dense Hidden Layer - ReLU activations
- Dense Hidden Layer- ReLu activations
- Dense Output Layer- Sigmoid activation

Each densely connected hidden layer contained 50 neurons. I experimented with many different configurations of model architecture and experimented with the use of Dropout layers to avoid overfitting and Batch Normalization layers, but this model produced the strongest results.

The model was compiled and trained with:

- Objective function: Custom (outlined above)
- Optimizer: Adam
- beta_1 = 0.9

- beta_2 = 0.99
- Learning rate: 0.001
- Batch size: 32
- Epochs: 100

The resulting scheme produced a vast improvement over the SGDClassifier and FiveThirtyEight predictions.

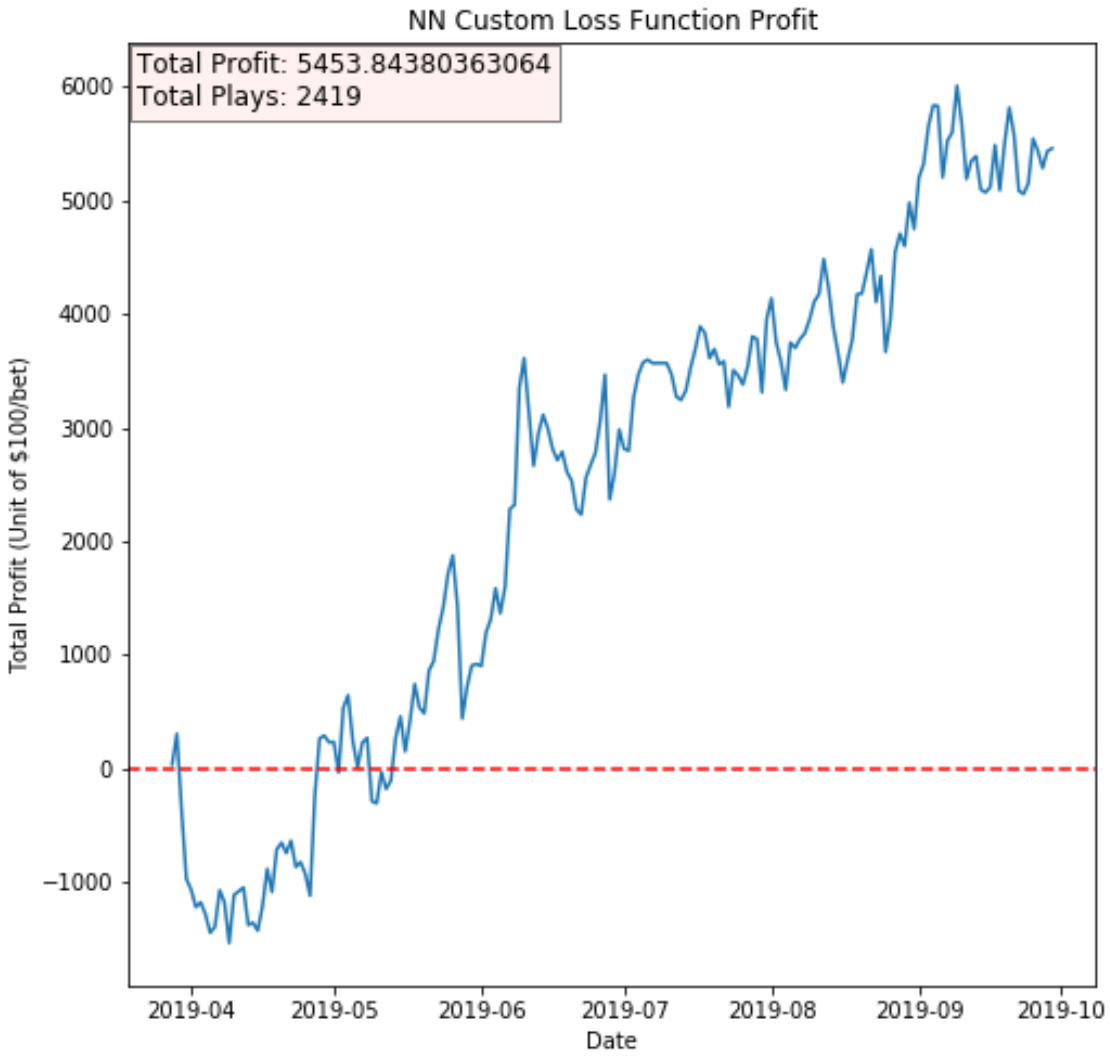

Figure 17: NNResults

Due to nature of the loss function (the ReLu function), the probabilities returned by the model were all either very close to 1 or 0. Thus, generating an arbitrage strategy as above was not possible. However, there was some additional success if only underdogs were played:

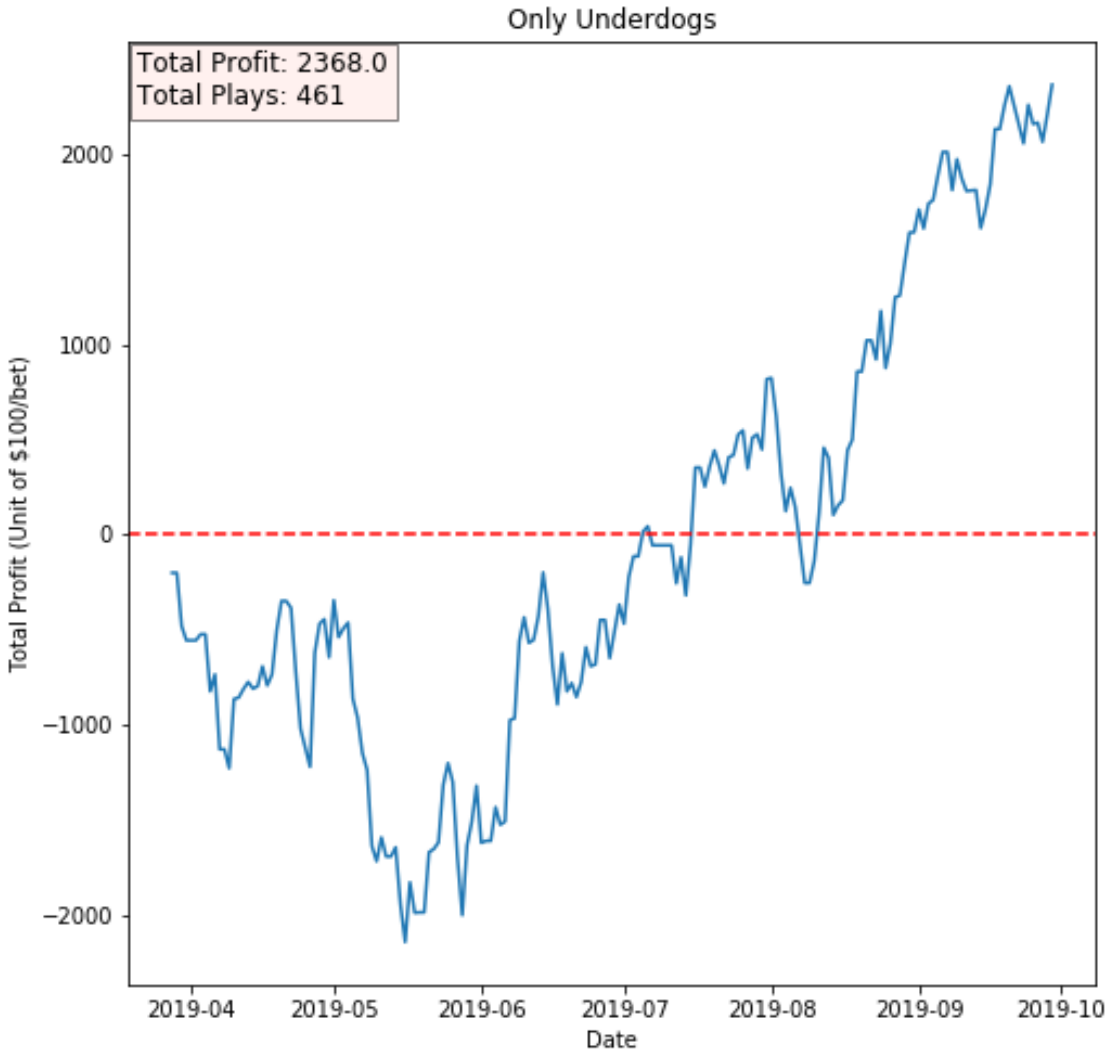

Figure 18: NNResults2

There is a third model in the main repo directory, an XGBoost model, that has shown flashes of success the success has not been repeatable.

Conclusion

The Neural Network trained on the entire feature set with the GaussRank scaled data is the only model that showed real promise. Using the scheme of betting on only home underdogs predicted that the model predicted to win, the model would have yielded a ROI (return on investment) of 12.67%. Using the model with the arbitrage strategy outlined, the model would have yielded a 5.43% ROI. Given enough volume, these are significant results. However, for the casual sports bettor, I do not recommend quitting your day job.

Vegas is not easy to beat, but this has at least shown that there are cracks that can be found.

There are several things to further this investigation:

1. Dimensionality reduction without information loss. Collinearity and multicollinearity are still a problem, even with reduced feature sets. Developing a team rating system, similar to Offensive rating and Defensive rating in the NBA, has the potential to incorporate more of the statistics in one feature.
2. Retraining the model throughout the season. Baseball seasons are not directly comparable; features were introduced to address era-based differences but as of the writing of this paper, retraining during season did not improve model results.
3. Looking deeper into the aribtrage strategy using the model predictions.
4. As does Vegas, every model built over-weighted home teams. Strategies attempted to correct this were:
    - Heavily weighting the ROAD WIN class (even though the classes are not imbalanced) to try to predict more road wins.
    - Heavily weighting underdogs
    - Using multi-label binary classification to try to establish confidence intervals.

    As of yet, none of these things have worked. But this will be an area of future research.

APPENDIX

References

Code Repository

- https://github.com/asoane34/retrosheet_analysis

Research and Data References

- ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/
- https://www.retrosheet.org/
- https://github.com/fivethirtyeight/data/tree/master/mlb-elo
- https://www.baseball-reference.com/
- https://sportsbookreviewsonline.com/scoresoddsarchives/mlb/mlboddsarchives.htm
- https://www.fangraphs.com/
- https://github.com/aldente0630/gauss_rank_scaler
- https://keras.io/