CSC311 Final Project Report
Predicting Major League Baseball Game Outcomes

Maxwell Bridgewater
Semyeong Hong
Department of Computer Science
University of Toronto

CONTENTS

CSC311 Final Project Report Predicting Major League Baseball Game Outcomes

I. Introduction

Baseball is one of the most popular sports in the world with approximately half a billion fans. Major League Baseball (MLB) is the world's best and most watched baseball league. In the last two decades the MLB has become dominated by numbers and statistics. Every event that can feasibly be measured in a baseball game is, and is done so meticulously. As a consequence of this wealth of information, the sport of baseball lends itself way to data analysis and machine learning techniques. Coaches, betting companies, and fans analyze the statistics in the hopes of gaining insight into future performance for a given team. In this report, we hope to provide the reader with a review of the existing literature on predicting the outcomes of MLB matches as well as our own contribution in applying machine learning methods to solving this problem.

II. Literature Review

As outlined in our introduction, baseball naturally lends itself to the field of data analysis and machine learning. Therefore the concept of applying machine learning techniques to predicting match outcomes is not new and has been attempted before. In 2015, Cserepy et al looked at every MLB match since 1921 and simulated individual games based on each team's overall statistics and other match specific variables [1]. Their approach was to Monte Carlo simulations with Markov decisions processes for each state seen in a baseball game. Their approach yielded an accuracy of 50.1% which they found would be profitable to bet on in the long run. In 2018, Elfrink evaluated the performance of linear models, random forests, and XGBoost in predicting the outcome of MLB matches [2]. They managed to achieve at best an accuracy of 55.5% in correctly predicting match outcomes. Elfrink mentioned limitations in their work with respect to computing power and time which meant that "5-fold cross-validation" could not be performed on their entire dataset which may leave room for improvement in their approach. In 2022, Li et al looked at the 2015-2019 seasons to try and predict match outcomes [3]. They examined the use of convolutional neural networks, artificial neural networks, support vector machines, and linear regression approaches for this task. Ultimately, they found that their SVM model performed the best with an average accuracy of 65%. They took the approach of creating a model for each of the thirty teams in the league, this allowed their model to better learn features specific to a particular team rather than more general features of any team. They also mention that their work uses team statistics which may reduce the contributions of individual players that may have been critical in a team's performance.

III. Problem Definition

We now seek to define the scope of the problem being tackled in this paper. To give a sense of the challenge of predicting MLB game outcomes we notice that in the 2022 season on average the best teams only won approximately 60% of their games and the worst teams won approximately 40% of the time. Clearly there are many factors that go into determining the outcome of a particular game. These can range from the obvious like a team's overall player statistics to when the game was played, weather on the day, and location of the game. The goal for this paper is to attempt to answer the following questions:

1) Can we build a classifier that reliably ($> 50\%$ accuracy) predicts the winning team for a given MLB game?
2) Can we construct a regression model that reliably predicts the total number of runs scored by each team playing in a particular MLB game?

To try and answer the above questions, we will use the following models. For the classification task, our basic model will be logistic regression, and the more sophisticated model being XGBoost acting as a classifier. For the regression task, we just want to determine whether the XGBoost model is suitable for the task of learning how to predict run outcomes. The regression task is an extension to the existing literature which primarily focuses on solving the classification problem to determine which team will win a given game. For this reason, our solution to the regression problem focuses on determining the validity of using an XGBoost regression model. Details on how this is implemented are outlined in the Models section of this report.

To train each the models listed above, we need to define the input vector that will be used. The following defines the parameters we use to construct the input vectors to our models:

- Team 1's name labelled with an integer
- Team 1's AVG
- Team 1's OPS
- Team 1's ERA
- Team 1's WHIP
- Team 1's total runs scored in the season at the point of the current game
- Team 1's Earned runs scored in the season at the point of the current game
- The above statistics for team 2
- Binary variable indicating whether team 1 is the home team

This amounts to a 15 dimensional input vector (for a complete description of the input variables see the appendix). In the context of the classification question, the target is the binary variable of whether team 1 wins or not. We indicate a win for team 1 with a 1 and a loss with a 0. For the regression problem, the target consists of two variables: the total number of runs scored by team 1 and team 2.

IV. Implementation

A. Data Collection

We initially utilized the official MLB API to gather data, employing Python's Request module for this purpose. The MLB's official API offers different endpoints for each type of collectible data. The endpoints we focused on were the team stats and schedule endpoints. First, we obtained key statistics for all 30 MLB teams from the team stats endpoint for every day of the 2022 and 2023 seasons. These key stats include batting average, OPS, ERA, WHIP, runs, and earned runs. These are important indicators of each team's performance in the season so far. Subsequently, from the schedule endpoint, we gathered details of games for the same periods. This data includes the date, names of the teams playing, and the final scores of the games.

After collecting this data, we stored it in a PostgreSQL database and proceeded with data polishing. We joined each game and its result with the corresponding team stats data based on the date and teams involved, creating inputs and targets. The final, organized inputs and targets were extracted as CSV files, ready to be used for model training.

We encountered a challenge in loading our desired data. We hypothesized that using the average individual stats of players who actually participated in the games as inputs would yield more precise results from our model. Unfortunately, the MLB's official API only provided averages for the 26-player rosters of each team and game, without any means to determine which of these players actually played in each game. So we were forced to use the average key stats of the 26-player rosters for each game and team (given the time constraints of this project), and our data includes these figures.

B. Models

We will begin by discussing the models used in the solving the classification problem outlined in this report. For the simple model, we chose to use Scikit's implementation of a logistic regression model. This was chosen because it is fast to train and a model with low complexity. We can also interpret the coefficients learned in a logistic regression model as the relative of importance that the model has learned from the data for each feature. We also decided to use the XGBoost model operating in a classification regime as a model with more complexity. We chose this model because we believe that the features that determine the outcome of an MLB game (that we can collect data for) most likely have complex non-linear relationships. XGBoost is one implementation of the family of models called gradient boosted decision trees. To break this down, gradient boosting is the process of minimizing a given loss function by using an ensemble of weaker learning algorithms before combining each of their predictions to get better performance overall. In the context of XGBoost, this model uses N decision trees of relatively low depth in combination with L1 and L2 regularization to quickly train accurate models with minimal computational resources. Where the 'XG' or extreme part of the name comes from the model's ability to perform computations in parallel during training time. Given the time and computational constraints of this project, a model like XGBoost that can function in both regression and classification schemes is a very suitable choice. For both the models used the primary metric for evaluating their performance is the accuracy of their predictions. We have labelled each of our input vectors with a 1 indicating a win for the home team and a 0 indicating a win for the away team. Therefore, if the model predicts at test time an output of > 0.5 then we interpret this as the model predicting a win for the home team, similarly outputs ≤ 0.5 are interpreted as a win for the away team.

In the context of the regression problem to determine whether our XGBoost regression model had learnt how to correctly assign runs scored by each team we decided to compare the model with a naive model. The naive model is extremely simple, it simply takes the average of the training data and uses these values as the runs scored by either the home or away team. The performance metric used in evaluating the regression models is the root mean square error (RMSE) metric. This was chosen because it is a standard way to evaluate regression models and assumes unbiased errors drawn from some normal distribution. If the XGBoost model achieves a smaller RMSE value compared to the naive model, then we can argue that the XGBoost regression model has learnt how to predict run outcomes.

C. Hyper-Parameter Tuning

One of the main challenges in using a complex model like XGBoost is the number of hyper-parameters available to be adjusted. One way to overcome this challenge is to use Bayesian Optimization to find the ideal set of hyper-parameters given the training data. To do this, we used the Hyperopt python library. The workflow for this process is as follows:

1) Create hyper-parameter search space
2) Define an objective function
3) Run the optimization

For the hyper-parameter search space, we chose to perform the optimization over the following hyper-parameters:

- Maximum depth of a decision tree.
- Gamma factor which controls the minimum loss needed to split a tree at a given node.
- L1 or 'alpha' regularization.
- L2 or 'lambda' regularization.
- The number of column samples by a tree. This controls the percentage of features from the input vector to be randomly used to build a given tree.
- Minimum child weight, this controls over fitting in the model.
- The number of trees that the model uses while performing the boosting step.
- The learning rate of the model.

The objective function for the optimization process is also a careful consideration. For the classification task we chose to minimize the negative of the accuracy of the model tested on the evaluation dataset. This corresponds to maximizing the accuracy of the model. Where as for the regression problem, we chose to minimize RMSE.

V. Results

A. Classification Problem

We begin by discussing the results from the logistic regression model. We did not perform Bayesian hyper-parameter optimization on the logistic regression model. This model was trained on the identical training set as the XGBoost model, and this was ensured by choice of the random seed. While training the model, we can visualize how the model is learning using the following learning curve.

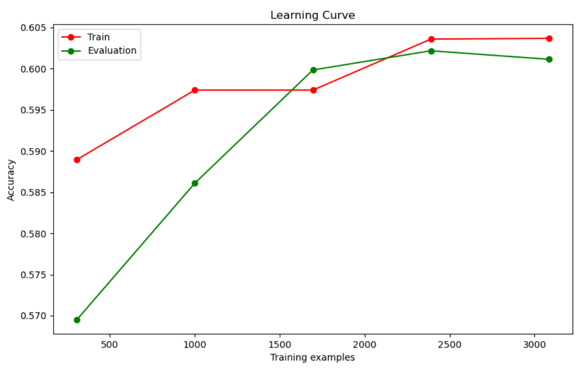

Fig. 1. Learning curve for binary logistic regression problem. We notice that the accuracy for the training data increases steadily as the model is given more examples to learn from. Whereas the evaluation set also improves with more data before plateauing and slightly decreasing.

Overall it was found that the **logistic regression model had a test accuracy of** 61.8%. Because the coefficients fit when performing binary logistic regression become weighted by the features the model found to be more useful in predicting game wins or losses, we can construct an importance plot.

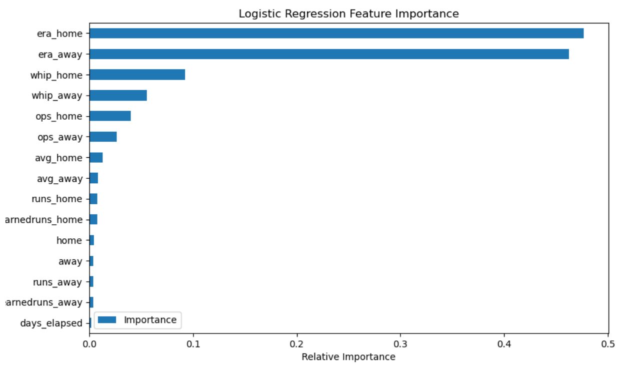

Fig. 2. Importance plot for binary logistic regression model.

Figure 2 shows that this model found the most important statistics to be the home and away teams' ERA values. In baseball, ERA stands for earned run average and quantifies the number of earned runs allowed by a team's pitcher over the course of a game. Earned runs refers to runs scored without an error occurring. These values are primarily used to judge the quality of a pitcher's play over the course of a game. It is interesting to note that the model found little importance given to which teams were playing, who was the home team, or when in the season the game was played.

We now consider the results from the XGBoost model. We can visualize the learning curve for the XGBoost model using a plot of RMSE against the number of estimators used.

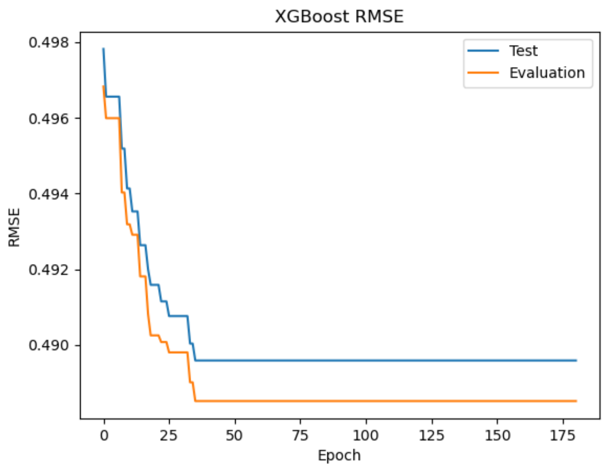

Fig. 3. Learning curve for XGBoost classification model. Here we see how the XGBoost model quickly decreases the loss function to a steady minimum in around $35-37$ training epochs. The training data curve does reach a lower minimum, but that is to be expected and the evaluation RMSE value is not far off.

Overall it was found that the **XGBoost model had a final test accuracy of** 58.8%. This result is interesting because it shows that for the classification task, the more complicated model that should be capable of better capturing non-linearity's in the data has a worse test accuracy compared to the more simple logistic regression model. Because we used a XGBoost model, it is still possible to generate an importance plot so we can determine what features the model found to be most useful. XGBoost feature plots are generated by ranking the number of times a given feature is used to split on a decision tree. Because XGBoost carries out this process automatically, it may determine that some features are unimportant and so are never used to split a node on a tree. Because of this, the feature importance plot may not contain all of the model parameters in the input vector.

Figure 4 shows us that the XGBoost model also found the team statistics like ERA and WHIP to be more important in the task of predicting game outcomes. The plot also tells us that only seven of our fifteen features are useful enough to the model to warrant a decision tree split on.

B. Regression Problem

We now discuss the results of the regression problem. By examining the training dataset's target vectors, it was found that the mean number of runs scored by the home and away team for any given game is: $y_{avg} = (4.443, 4.447)$. **This resulted in an RMSE value when compared with the test**

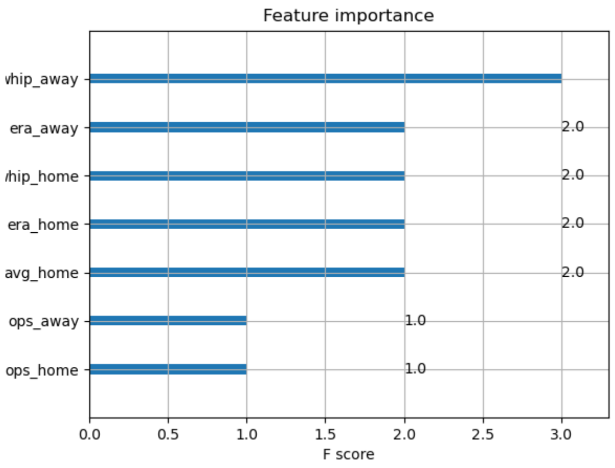

Fig. 4. Importance plot for XGBoost classifier.

set of 3.227. This is the target for which we want the XGBoost regression model to be below.

Moving onto the results from the XGBoost regression model. We can again visualize the learning curve using the RMSE value at different training epochs.

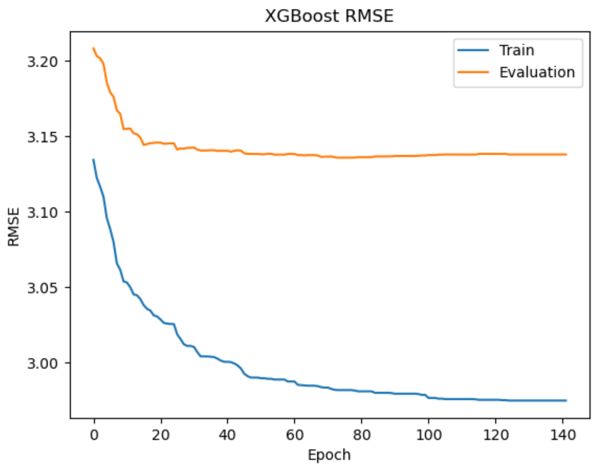

Fig. 5. Learning curve for XGBoost regression model. In the regression context, it takes over 100 epochs for the model to reach a minimum RMSE value. The training data curve does reach a lower minimum, but that is to be expected and the evaluation RMSE value differs by approximately 0.2.

Overall, the final RMSE value of the XGBoost regression model when compared to the test data was found to be 3.135. Although the decrease was marginal, it does indicate that the model learnt how to predict run outcomes better than just the average the data it was trained on. However, it is a marginal improvement. Given that RMSE values are in the same units as the target vector, our model would have an error of around 3 runs when trying to predict game outcomes. Despite this, we can still gain information about which features the model found to be important with an importance plot.

Figure 6 shows that the regression model still placed importance on the specific team statistics. However, this model also placed a large amount of importance on the home and away teams. Interestingly, the home team matter significantly more than the away team. This indicates a possible bias or advantage to the team playing at their home stadium. We also tried to modify an approach in the literature that gave models the date the game was played by changing this feature into the days elapsed since the season had started. However, it seems that in both the classification and regression models this feature had little to no importance on the model's predictive ability.

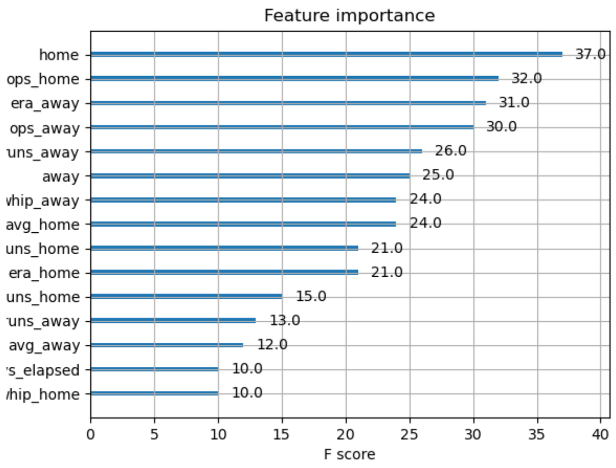

Fig. 6. Importance plot for XGBoost regression model.

VI. Conclusion

In conclusion, we found that our work definitely agrees with the existing literature on the subject. We achieved prediction performance for the classification task in line with the quoted $55-65\%$ accuracies found by other researchers. We found that for the classification task, a simple logistic regression model outperformed the more complex XGBoost classifier. This indicates that either the problem is more linear than originally thought or our implementation of the XGBoost classifier was flawed. It should be noted that we only considered data from the 2022 and 2023 seasons, so perhaps by using data from previous seasons we could see improvements in the XGBoost classifier's accuracy. For the regression task, we found that the XGBoost regression model did succeed in showing that it was capable of learning from the data available. However, it did not improve much on the naive model previously discussed. A major limitation of this work is that we only used data from two seasons, perhaps with more data available to use we can improve the regression model's predicting capabilities. Overall, the problem of predicting baseball outcomes remains extremely challenging; but the authors believe that classification accuracies can be improved with a careful approach to data prepossessing and model architecture.

Appendix A
Definition of Baseball Statistics

- Batting average: Batting average is calculated by dividing the total number of hits a batter has made by the number of at-bats they have had.

- OPS: OPS combines a batter's on-base percentage and slugging percentage. The on-base percentage is calculated by dividing the total number of times a batter has reached base by the number of at-bats they have had, and the slugging percentage is calculated by dividing the total number of bases a batter has hit for by the number of at-bats.
- ERA: The Earned Run Average (ERA) is a metric that reflects a pitcher's performance. It is calculated by dividing the number of earned runs allowed by the pitcher by the number of innings pitched and then multiplying this figure by 9. This represents the number of earned runs the pitcher is expected to allow over an average of 9 innings.
- WHIP: WHIP (Walks plus Hits per Inning Pitched) is a statistical measure of a pitcher's efficiency. It is calculated by dividing the sum of walks and hits allowed by the pitcher by the number of innings they have pitched. WHIP indicates how many base runners a pitcher allows per inning on average, with a lower WHIP indicating better performance.

Appendix B
Link to Github

Link to Github page.

References

[1] Nico Cserepy, Robbie Ostrow, and Ben Weems. *Predicting the Final Score of Major League Baseball Games*. 2015.

[2] Tim Elfrink. "Predicting the outcomes of MLB games with a machine learning approach". In: *Vrije Universiteit Amsterdam* (2018).

[3] Shu-Fen Li, Mei-Ling Huang, and Yun-Zhi Li. "Exploring and selecting features to predict the next outcomes of MLB games". In: *Entropy* 24.2 (2022), p. 288.