Predicting Winners: Using Regression Models to Explore Baseball Performance

Alex Ramponi
University of Texas - Austin
alex.a.ramponi@gmail.com

Contents

Abstract

Part of what makes baseball unique compared to other sports is the abundance of statistics to describe a player's performance. While the stats on a player's baseball card ultimately do not determine a winner, the abundance of data from 162 regular season games paint a good picture about that player's performance on average. When you factor in 30 teams in the league, we can start to extract trends about player performance and what impacts a team's chances of winning on a daily basis. Using exclusively the MLB Stat API (MLB, 2024), I was able to determine the winner of games at an above-average 60% success rate using different machine learning models based on XGBoost and Neural Networks.

1 Introduction

In baseball, the ultimate objective is to score more runs than your opponent. A run is scored when the batter gets on base as a runner (by either a walk or a hit off the opposite pitcher) and eventually passes home plate. Although defensive teamwork is important to get runners out on base, at the end of the day baseball is a duel between the pitcher and the batter. Thanks to each player's large sample of games in a season, we can create a dataset to predict how many runs a team scores based on their own batters and the opponent pitcher. The goal for this project is creating a model that predicts both team's runs scored and chooses a winner based on who scores more runs. To create the best model for this, I wanted to explore popular machine learning regression models - **XGBoost** and **Neural Networks**.

2 Dataset Feature Selection and Gathering

2.1 Gathering Data - The MLB API

All of the baseball data in this model comes directly from the MLB API (MLB, 2024). The MLB API has a large amount of HTTP endpoints available to query data. However, this API usage is made significantly easier thanks to the MLB Stats API Python package (Roberts, 2024). I used this package exclusively and got my starting point and inspiration for it from the blog post by Adrien Pelzer (Peltzer, 2023). After getting our data pipeline setup, it was time to explore the options available for each of the players.

2.2 Choosing which games to use

In Baseball, it is common belief that the postseason is unpredictable. Not only is it a common superstition that the team that wins is whoever is "hot", but there is actual research showing that star player performance tends to drop noticeably during the postseason (Conforti et al., 2021).

Because of this, I did not want to include any postseason data in the training or factor this into the model performance. Instead, this model would focus exclusively on games from April 1st to September 30th which is a bulk of the regular season games.

Additionally, I had to consider what years worth of data to use. The first consideration is that I should not include the year 2020 as this was when the COVID-19 pandemic was taking place. This season alone was very unique in that it was only 60 games, primarily divisional opponents, and all fans were removed from the game (Castrovince, 2020).

For many, it feels like the world changed after the COVID-19 pandemic. While there is not much clear data that baseball is significantly different pre-COVID and post-COVID, I decided to only include game data from 2021 onward. I personally felt as this data was the best equipped to predict future games as this has modern day technology, analytics, and baseball analysis to help teams perform.

Overall, I pulled player stats for the 2021, 2022, 2023, and 2024 seasons. For this project, I analyzed the data in two fashions which will be talked about in later sections:

- Train on 2021, 2022, 2023 and Evaluate on 2024
- Do a dataset split of **80%** training and **20%** evaluation, which has been shown to be one of the optimal split ratios based on the dataset size in the paper by (Muraina, 2022)

2.3 Team and Player Data

From my own experience as a fan of baseball and playing fantasy sports, my hypothesis is that a player's impact on the team's performance can be measured by the following four categories.

- Season stats
  - Using season stats, we can see how a given player is doing over the course of the full year. This is typically a better indicator than career stats due to changes

in performance due to age, team environment, or many other factors.

- Recent calendar date stats
  - The recent date stats are very important because they can show if a player is "hot" or "cold". For both pitchers and batters, they go through phases in the season where they'll be performing at a high level to performing terribly. The performance over the last few days has a huge impact on the projections and is almost more important than the season stats. The recent calendar date stats also show how much rest a player has had, specifically around pitchers. If a pitcher has played multiple games recently, they are more likely to perform worse. Therefore, this is a very important part of the player data.
  - Throughout this project, I chose the recent calendar day count to be **5 days back**. As an example, for a game on April 10th, we will consider the sum of the stats over: [April 5th, April 6th, April 7th, April 8th, April 9th]
- Stats of the pitcher they are facing
  - For a team to score runs, they need to get on base which all starts based on the opposing team's pitcher performance. A poorly performing pitcher typically indicates that the batters will do better. The opposite is usually true where a highly performing pitcher will cause the batters to perform worse and therefore, score less runs.
- The batting order lineup
  - As shown in the blog post by Dr. Randal S. Olson, batting order is very important as batters towards the top of the lineup have more at bats and more changes to get on base (Olson, 2018). Additionally, having certain batters before or after each other can create rippling effects on runs scored. For example, if you have a batter who is really good at getting on base (either a walk or a hit) and after them you have a batter who is even better at slugging (getting hits for multiple bases at a time, typically via home-runs or deep hits), combining these together can cause more runs to be scored. Therefore, the order of the batter lineup is crucial for a team's overall performance.

2.4 Feature Set

Using these categories, I decided to combine all of these data points as a team to create the feature set (independent variable) to measure against the runs score (dependent variable). The feature set looks like the following pseudoscope:

```
lineup = [player_0, ... player_8]
opposite_pitcher = player_pitcher
f = [] # team features
for batter in lineup:
    f = f + batter.season_stats
    f = f + batter.last_x_date_stats

f = f + opposite_pitcher.season_stats
f = f + opposite_pitcher.last_x_date_stats
```

For the statistics above, I used every stat the MLB API offered for each type of data. There were two categories of fields — pitching and batting. I have listed the fields used in Appendix A for pitching and Appendix B for batting. For each of the individual player statistic categories (season stats and last 5 day stats), we used the same features seen in the Appendix.

- For pitchers, we had a total of 61 features per category - F_p
- For batters, we had a total of 34 features per category - F_b

In total, we had the following number of features:

$$\text{Total Features} = 9 \times (2 \times F_b) + (2 \times F_p)$$

Substituting the values:

$$\text{Total Features} = 9 \times (2 \times 34) + (2 \times 61) = 734$$

Leaving us with 734 features to predict the singular label of total runs score.

2.5 Dataset statistics

As discussed previously, we are pulling the 734 features and 1 label for all regular season games between April 1st and September 30th in 2021, 2022, 2023, and 2024. Excluding 5 games that there was no API data available for, this resulted in data from

9,791 games. Because each game contains two teams with a label each (runs the team scored), this resulted in a total dataset size of **19,582 feature datasets**. For 734 features and 1 label for each, this resulted in approximately 14,392,770 values with a size of **64MB**.

3 Neural Network Model

The first model I explored was using Neural Networks. Deep learning and Neural Networks are breakthrough technologies when looking at a modern history of AI (Schmidhuber, 2022). Deep learning networks work well on large dataset training and they can learn complex relationships between features in data (Cloud, 2024). Because of these benefits, I wanted to explore their capability to doing linear regression on our dataset to predict runs scored.

3.1 Model Design

To create the initial model, I used the linear layer from the PyTorch library. To have the neural network work as a universal approximator (Augustine, 2024), I needed to include a non-linear activation layer. Referencing the comprehensive study and benchmark of activation functions, I chose to go with the ReLU function as this was deemed to be the most effective to train and work with in most network styles (Dubey et al., 2022). For the initial network design, I needed to determine the number of hidden layers to use in the network. I referenced the work done by Uzair et. all who recommend three hidden layers as a good middle ground that balances accuracy and over-fitting (Uzair and Jamil, 2020). I also referenced the instructions in the Medium post by Krishnan to determine the size of these hidden layers should be [470, 350, 200], with 470 being the first hidden layer and 200 being the final hidden layer which results in a linear output of size 1 for the number of runs (Krishnan, 2021).

Now that we have the layer design, I included Batch Normalization to the network. Batch Normalization gives the network quicker convergence and less reliance on the initial weight values, ultimately allowing for more robust training and potentially better performance (abhilashgaurav003, 2024). With this layer setup, we are in a good position to begin training and fine tuning our model and hyperparameters.

3.2 Training The Model

With the initial model structure setup, I had to then look into the best way to train the model. Because the goal is to predict the number of runs scored given the team's features, I used the Mean-Squared-Error Loss function

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

To optimize the model, I referenced the work done by Zhou et. All to determine that SGD (Stochastic Gradient Descent) tends to generalize better than ADAM in Deep Learning. (Zhou et al., 2021) Because training speed is not a major concern, I wanted this better generalization so I chose to use the SGD optimizer for all of my model training.

In terms of dataset training, I took all of the years of data and did a random split on training and evaluation data instead of focusing on training on certain years and evaluating on others. This would allow me to tune the hyper-parameters in an unbiased way and not optimize for a certain year. For the ratio, we continued to use the same ratio referenced above of an 80% training, 20% evaluation split (Muraina, 2022).

3.3 Fine Tuning

With the training setup and initial model established, it was then time to fine tune the model structure and training hyper parameters to determine the best possible model.

3.3.1 Layer Count and Size

We initially guessed the optimal layer count and size based on the provided literature to be [470, 350, 200]. However, I wanted to validate that this was infact the best possible hidden layer setup. To check this, I evaluated multiple sets of hidden layers on the following training hyper parameters:

- Epochs: 100
- Learning Rate: 0.01
- Optimizer: SGD
- Batch Size: 256
- Dataset Split: 80% training, 20% evaluation

As we can see in Table 1, the literature recommended layer size performed better compared to

Layer Size	**Training MSE**	**Evaluation MSE**
[470, 350, 200]	4.6367	6.8656
[2048, 1024, 512, 256]	2.3131	7.4590
[256, 512, 1024, 2048]	4.4390	12.6236
[256, 512, 1024, 2048, 4096]	3.2465	7.6633
[256, 512, 1024, 2048, 4096, 8192]	2.1890	9.0332

Table 1: Results of different layer sizes and layer counts on the model performance

larger models. The larger models had better training accuracy, but failed to achieve a better evaluation loss, implying some form of overfitting was occurring. Overall, we can assume that using [470, 350, 200] will serve our model's generalization well.

3.3.2 Optimizers and Step Size

With the layer size being determined, we can then evaluate the optimizer we use. Initially, we determined that SGD is the best due to it's generalization ability (Zhou et al., 2021). However, the dataset makes an impact on the correct optimizer so will still explore alternatives. The learning rate is closely tied with the optimizer we use so it is best to test these together. To see what the best setup is for generalization, we will test combination of optimizers and step sizes on the following:

- Batch Size: 256
- Epochs: 100
- Layers: [470, 350, 200]
- Dataset Split: 80% training, 20% evaluation

As seen in Table 2, the results are shocking clear that the ADAM optimizer strongly outperforms the SGD optimizer. This is likely due to the ADAM Optimizer converging faster than the SGD optimizer thanks to ADAM's ability to do coordinate-wise clipping on the update of the parameters (Pan and Li, 2023). Additionally, the optimal generalization step size from ADAM was 0.01 with the default Betas being used of (0.9, 0.999) for all ADAM instances.

Optimizer	**Step Size**	**Training MSE**	**Evaluation MSE**
ADAM	**0.01**	0.8911	5.6966
ADAM	**0.005**	0.8465	7.4032
ADAM	**0.001**	0.8074	6.1794
SGD	**0.01**	4.6108	8.7716
SGD	**0.005**	4.6259	7.0405
SGD	**0.001**	7.7030	8.2744

Table 2: Evaluation of SGD vs ADAM and their respective step sizes on the MSE Loss

3.3.3 Summary of Model Hyper-parameters

Given the above sections, we can now finalize the following Neural Network Model parameters:

- Hidden Layers: [470, 350, 200]
- Training Batch Size: 256
- Optimizer: ADAM with default betas of (0.9, 0.999)
- Optimizer Step Size: 0.01
- Overall Size: 2.3 MB

In **Section 5** we will analyze the model more specifically around the task at hand of predicting baseball game winners.

4 XGBoost Model

I initially assumed Neural Networks were the best tool for this task, but after doing some research I quickly learned that XGBoost is not only very popular, but incredibly powerful for regression tasks thanks to it's tree boosting system (Chen and Guestrin, 2016). Firstly, there is a considerable amount of research looking into why XGBoost outperforms Neural Networks on tabular data, specifically on datasets containing around 10,000 examples similar to the dataset used in this project (Grinsztajn et al., 2022). This research finds after many experiments that XGBoost remains state of the art despite the promising potential of Neural Networks. Additionally, XGBoost is significantly faster to train and evaluate on. Because of this promising potential, I wanted to explore XGBoost applied to the baseball winner problem.

4.1 Model Design

Having never worked with XGBoost in the past, I had to first learn the libraries for training and evaluating this model. I found the XGBoost introduction from Kaggle incredibly helpful as it showed how to create a model and more importantly, how to tune the parameters for it (Ellis, 2022). Using this, I was able to create an initial baseline model and begin tuning hyper-parameters.

4.2 Tuning Hyper-parameters

While the XGBoost model is very easy to quickly get up and running, the magic lies in the hyper-parameters used to create the model. These dictate everything from how many tree estimators to use, how deep each estimator should be, learning rate, and many other factors. I followed a very helpful guide to determine the most impactful values and what they do for the model (RITHP, 2023). For my initial tuning, I did an incredibly aggressive GridSearchCV (Okamura, 2020) search over the following space of parameters:

`max_depth`: {4, 5, 6, 7, 8, 9, 10}
`n_estimators`: {300, 400, 500, 600, 700, 800, 900, 1000}
`learning_rate`: {0.1, 0.05, 0.01}
`subsample`: {0.6, 0.7, 0.8, 0.9, 1}
`colsample_bytree`: {0.6, 0.7, 0.8, 0.9, 1}

Using a cross-fold of 3 to help evaluate our training data without overfitting, this results in 12,600 different model hyper parameter combinations. This hyper-parameter GridSearch took over 24 hours but resulted in the following values:

`max_depth`: 5
`n_estimators`: 1000
`learning_rate`: 0.01
`subsample`: 0.6
`colsample_bytree`: 0.9

With these values, we then continued to narrow down and try different hyper-parameters categories and values. Specifically, we tried increasing and decreasing values in ranges around the original value. For example, using n_estimators of {900, 1000, 1100, 1200, 1300, 1500, 1700}. Additionally, different values like the following were included in the GridSearch:

`gamma`: {0, 0.1, 0.5, 1}
`min_child_weight`: {1, 3, 5}
`reg_alpha`: {0, 0.01, 0.1}
`reg_lambda`: {1, 10, 100}
`grow_policy`: {depthwise, lossguide}

Ultimately, this resulted in the final hyperparameters:

`tree_method`: hist
`colsample_bytree`: 0.9
`gamma`: 0.5
`grow_policy`: depthwise
`learning_rate`: 0.01
`max_depth`: 5
`min_child_weight`: 1
`n_estimators`: 1500
`reg_alpha`: 0.01
`reg_lambda`: 1
`subsample`: 0.6

4.3 Trained Model Statistics

While we will do more in-depth evaluation of the model's performance in **Section 5** around the actual Baseball game prediction, it is still helpful to evaluate the model on it's direct regression output. For our model, we achieve the following results **on an evaluation dataset** for a full dataset split into 80% training data and 20% evaluation data:

- $R^2 = 0.6001$
- $MSE = 3.835$

5 Evaluating Performances

For our task, ultimately the goal is to predict the winner of a game. To do this, we want to compare the predicted runs for the home team vs the predicted runs of the away team and see who is predicted to score higher. However, before diving into that, we will first take a look at the traditional regression evaluation metrics to measure performance this way.

5.1 Model Regression Performance Evaluation

To evaluate the regression performance, we used the entire dataset for both models and did a random split of 80% to 20% to give us our training and evaluation datasets respectively.

Using our best Neural Network model from **Section 3.3.3** and our best XGBoost model from **Section 4.2**, we have the following R^2 and MSE values

Model	R^2	MSE
Neural Network	0.398	5.982
XGBoost	0.600	3.835

.

As we can see, the metrics for XGBoost are nearly twice as accurate as the Neural Network. This shows the Neural Network struggled to predict the actual runs much more than the XGBoost model. However, in the next section we will see if there is as noticeable as a difference in accuracy.

5.2 Model Game Accuracy Evaluation

When evaluating the actual game accuracy, we have to rethink our data-split from the regression metric analysis. Instead of doing the 80:20 split of our total dataset, we instead want to split based on the years of game data.

Therefore, when evaluating the accuracy of a model, we will focus on a year. When evaluating a year, the model will have been trained on the other years. For example, if we are evaluating on 2024, we will have trained the model on 2021, 2022, and 2023. If we are evaluating on 2023, we will have trained on 2021, 2022, and 2024. The accuracy of the models are below

Model	2021	2022	2023	2024
Neural Network	55.13%	57.70%	56.04%	56.88%
XGBoost	60.62%	61.12%	60.68%	61.39%

As we can see, there is a noticeable performance difference between the XGBoost model and the Neural Network model. This was also seen in the regression evaluation in the R^2 and the MSE metrics. **We can confidently conclude that XGBoost model outperforms the Neural Network on our problem space**.

5.3 Feature Importance

Now that we concluded our XGBoost model performs best for our task, we will exclusively analyze this model from this point forward. To trust and understand the model, it is important to understand why it predicts the values it does. Models should not be designed as black boxes, but designed as interpretable models so we can understand the outputs (Rudin, 2019). To build this trust in a model, we can use a few methods - SHAP and LIME - which can be easily applied to XGBoost models following the guides such as the one seen here (ten Heuvel, 2023).

5.3.1 SHAP Analysis

SHAP values (SHapley Additive exPlanations) is a method based on cooperative game theory and is used to increase the transparency and interpretability of global and local model values (Trevisan, 2022). The global interpretability can be found by aggregating the SHAP results for a training set to observe the general trends. Additionally, SHAP has been found to perform better when working with more sparse datasets which is the case for our baseball dataset (Roberts et al., 2022). For our baseball model, we will use SHAP to determine what typically influences the output based on our evaluation predictions.

The SHAP results for the top-20 results can be seen in the **Appendix Section C**. To summarize their results, the most impactful element was the Opponent Pitcher Last 5 Days Runs Scored. The higher this value was, the higher the predicted result will be. Additionally, the lower this value was, the lower the final result would be. This is a logical most impactful feature as this value determines the pitchers past games over the 5 days before a matchup. If this pitcher tends to give up a lot of runs, there is a good chance the team will score a lot of runs on this pitcher. Additionally, the pitcher last 5 days wins was the second highest value. The higher win count this pitcher has, the lower the results will be which also makes sense - if a pitcher wins, they typically did not give up many runs.

Overall, the SHAP analysis heavily favorites the last 5 days of stats, proving that a team's expected runs typically depends on how many runs they have scored in the days leading up to a game. An interesting observation from the SHAP values is that the bottom of the lineup (typically batters 6, 7, 8, and 9) show up for more stats than the top of the lineup. The top of the lineup in a baseball team is typically the "stars" of a team and the weaker hitters are typically towards the bottom. This is further shown by the evidence that a batting lineup makes a difference in the overall team performance (Olson, 2018). While the bottom lineup players scoring runs tends to lead to a stronger performance, the SHAP plot in Appendix C also shows that as the 8th and 9th hitters play in more games, the total runs actually decreases. This indicates that a weaker batter in these slots makes a noticeable difference in the team's performance.

5.3.2 LIME Analysis

LIME is an explanation technique that learns an interpretable model locally around the prediction. (Ribeiro et al., 2016). Unlike SHAP, LIME is best utilized to explain a single prediction's result and understand what led to the output (Radečić, 2020).

For our LIME results, we will analyze the prediction for a specific game in the 2024 season evaluation dataset. Specifically, we will look at the Los Angeles Dodgers vs. Houston Astros from July 27th, 2024. This game was a thriller with the Astros winning 7 to 6 thanks to a late walk off home run. However, the model predicted this as the Astros losing to the Dodgers 4.15 to 6.53.

The LIME results for each team can be seen in the Appendix in section D. However, we can summarize the predictions below:

```
Astros Prediction Explanation:
pitcher_last_5_day_runsScoredPer9: 1.0859
8_bat_last_5_day_runs: 0.1710
9_bat_last_5_day_runs: 0.1139
8_bat_last_5_day_rbi: 0.0812
5_bat_last_5_day_runs: 0.0556
3_bat_season_intentionalWalks: -0.0330
6_bat_last_5_day_grdIntoDblPlay: -0.0299
5_bat_season_caughtStealing: 0.0273
4_bat_season_sacFlies: 0.0249
5_bat_last_5_day_grdOutsToAirouts: 0.0089
```

```
Dodgers Prediction Explanation:
pitcher_last_5_day_runsScoredPer9: 1.4976
8_bat_last_5_day_runs: 0.2594
9_bat_last_5_day_runs: 0.2461
pitcher_last_5_day_era: 0.1814
7_bat_last_5_day_runs: 0.1423
3_bat_last_5_day_numOfPitches: -0.0923
3_bat_last_5_day_atBats: -0.0827
6_bat_last_5_day_numOfPitches: -0.0710
3_bat_last_5_day_plateAppearances: -0.0568
1_bat_season_intentionalWalks: -0.0170
```

Similar to the results from SHAP, LIME results in a heavy weighting of the runs over the past few days by the pitcher and batter. A new observation from LIME is the appearance of the season intentional walks showing up. For the Astros, their #3 spot batter is Yordan Alverez who is an All Star player and a notorious home run hitter. Additionally, the Dodgers #1 spot batter is Shohei Ohtahni who is the 2024 National League MVP. Intentional walks are commonly used to manage the flow of the game and to shut down strong opponent batters to prevent big hits during inopportune times (Cameron, 2013). LIME being able to determine that the best player from each lineup being intentionally walked will hurt the team's performance demonstrates the models generalization ability as well as LIME's strength over SHAP to find details like this on a game by game basis.

Overall, it makes sense that the model was unable to predict this outcome as the starting pitchers did not determine the result but the bullpen. This is a missing feature-set from our model and it's weakness can be shown by this misclassification.

5.3.3 Feature Importance Summary

Overall, we can see that both LIME and SHAP strongly signal that the runs scored or given up over the past 5 days have the largest impact on a team's chance of winning today. This makes sense and it is interesting to see how much strongly the last 5 day stats are weighted compared to the season long stats for a team. This demonstrates how important trends are in baseball and how a team being "hot" truly matters.

6 Future Improvements and Use-Cases

Overall, the model performed above average and sets up a good baseline for further analysis. The model also helps reveal interesting insights into what makes a team perform via the LIME and SHAP analysis methods.

With the rise and legalization of sports betting in more states, this model could also be fine tuned further to attempt to out perform the popular betting platforms on categories such as winner of a game, player hits, and pitcher earned runs. There is interesting research done into this which suggests using other models such as Support Vector Machines (SVM) as well as pulling in more real time data such as weather of the playing field (Galekwa et al., 2024).

Additionally, there could be some improvement on shrinking the size of the dataset. We currently pull in every possible value available from the MLB Stats API and we did not remove any fields. However, statistics like the triple play are uncommon in baseball and will be 0 for an entire season for the vast majority of players. Values like these could be removed from the dataset to help speed up data collection and training. It is even possible to use the Shapley values from the SHAP analysis to select features which has been shown to significantly outperform other state of the art methods (Sebastián and González-Guillén, 2024).

Lastly, we could improve performance by including more baseball specific information into our feature-set. For one, we could include the matchup statistics of a batter vs. the opponent pitcher over both of their careers. This is an important stat that is available as it shows how well someone tends to do vs. a starting pitcher. On top of this, we can also include information about the opponent team's bullpen into the dataset. As of right now, the model does not consider the other pitchers on the opposite team outside of the starting pitcher.

Lastly, we could also include fielding statistics for the opposing team. Intuition says that a team of bad fielders would give up more runs than a team with good fielders. However, it is interesting to see in the blog post by Matt Wyatt that this is not the case and he found little to no significance of fielding performance on a team's chance of winning (Wyatt, 2024). Nonetheless, it would be a good area of investigation to add these values and measure model performance.

In summary, this model performed well and it was interesting to see the pros and cons of different machine learning models and methods to predict the teams runs. There are still many exciting possibilities to improve the performance of predictions and have it ready for the upcoming MLB season in 2025.

References

abhilashgaurav003. 2024. Batch normalization implementation in pytorch. URL: https://www.geeksforgeeks.org/batch-normalization-implementation-in-pytorch/.

Midhun T Augustine. 2024. A survey on universal approximation theorems. URL: https://arxiv.org/abs/2407.12895, arXiv:2407.12895.

Dave Cameron. 2013. What actually happens after an intentional walk? URL: https://blogs.fangraphs.com/what-actually-happens-after-an-intentional-walk/.

Anthony Castrovince. 2020. Faq: All you need to know about 2020 season. URL: https://www.mlb.com/news/faq-for-2020-baseball-season.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM. URL: http://dx.doi.org/10.1145/2939672.2939785, doi:10.1145/2939672.2939785.

Google Cloud. 2024. What is deep learning? URL: https://cloud.google.com/discover/what-is-deep-learning.

Christian M. Conforti, Ryan L. Crotin, and Jordan Oseguera. 2021. An analysis of playoff performance declines in major league baseball. *The Journal of Strength & Conditioning Research*, 35(12S):S36–S41. URL: https://journals.lww.com/nsca-jscr/fulltext/2021/12002/an_analysis_of_playoff_performance_declines_in.6.aspx.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. URL: https://arxiv.org/abs/2109.14545, arXiv:2109.14545.

Carl McBride Ellis. 2022. An introduction to xgboost regression. URL: https://www.kaggle.com/code/carlmcbrideellis/an-introduction-to-xgboost-regression.

René Manassé Galekwa, Jean Marie Tshimula, Etienne Gael Tajeuna, and Kyamakya Kyandoghere. 2024. A systematic review of machine learning in sports betting: Techniques, challenges, and future directions. URL: https://arxiv.org/abs/2410.21484, arXiv:2410.21484.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on tabular data? URL: https://arxiv.org/abs/2207.08815, arXiv:2207.08815.

Sandhya Krishnan. 2021. How do determine the number of layers and neurons in the hidden layer? URL: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3#:~:text=The%20number%20of%20hidden%20neurons,size%20of%20the%20input%20layer.

MLB. 2024. Mlb stats api. URL: https://statsapi.mlb.com.

Ismail Muraina. 2022. Ideal dataset splitting ratios in machine learning algorithms: General concerns for data scientists and data analysts. URL: https://www.researchgate.net/publication/358284895_IDEAL_DATASET_SPLITTING_RATIOS_IN_MACHINE_LEARNING_ALGORITHMS_GENERAL_CONCERNS_FOR_DATA_SCIENTISTS_AND_DATA_ANALYSTS.

Scott Okamura. 2020. Gridsearchcv for beginners. URL: https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee.

Dr. Randal S. Olson. 2018. Does batting order matter in major league baseball? a simulation approach. URL: https://randalolson.com/2018/07/04/does-batting-order-matter-in-major-league-baseball-a-si

Yan Pan and Yuanzhi Li. 2023. Toward understanding why adam converges faster than sgd for transformers. URL: https://arxiv.org/abs/2306.00204, arXiv:2306.00204.

Adrien Peltzer. 2023. Accessing mlb statistics using python. URL: https://medium.com/@adrienpeltzer_17089/accessing-mlb-statistics-using-python-e5b539985a96.

Dario Radečić. 2020. Lime vs. shap: Which is better for explaining machine learning models? URL: https://towardsdatascience.com/lime-vs-shap-which-is-better-for-explaining-machine-lea

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier. URL: https://arxiv.org/abs/1602.04938, arXiv:1602.04938.

RITHP. 2023. The main parameters in xgboost and their effects on model performance. URL: https://medium.com/@rithpansanga/the-main-parameters-in-xgboost-and-their-effects-on-mod

Claudia V. Roberts, Ehtsham Elahi, and Ashok Chandrashekar. 2022. On the bias-variance characteristics of lime and shap in high sparsity movie recommendation explanation tasks. URL: https://arxiv.org/abs/2206.04784, arXiv:2206.04784.

Todd Roberts. 2024. Mlb statsapi wiki github. URL: https://github.com/toddrob99/MLB-StatsAPI/wiki.

Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. URL: https://arxiv.org/abs/1811.10154, arXiv:1811.10154.

Juergen Schmidhuber. 2022. Annotated history of modern ai and deep learning. URL: https://arxiv.org/abs/2212.11279, arXiv:2212.11279.

Carlos Sebastián and Carlos E. González-Guillén. 2024. A feature selection method based on shapley values robust for concept shift in regression. *Neural Computing and Applications*, 36(23):14575–14597. URL: http://dx.doi.org/10.1007/s00521-024-09745-4, doi:10.1007/s00521-024-09745-4.

Thomas ten Heuvel. 2023. Opening the black box of machine learning models: Shap vs lime for model explanation. URL: https://medium.com/cmotions/opening-the-black-box-of-machine-learning-models-shap-vs-lime-for-model-explanation-d7bf545ce15f.

Vinicius Trevisan. 2022. Using shap values to explain how your machine learning model works. URL: https://towardsdatascience.com/using-shap-values-to-explain-how-your-machine-learning-model-works-732b3f40e137.

Muhammad Uzair and Noreen Jamil. 2020. Effects of hidden layers on the efficiency of neural networks. *2020 IEEE 23rd International Multitopic Conference (INMIC)*, pages 1–6. URL: https://api.semanticscholar.org/CorpusID:231682198.

Matt Wyatt. 2024. Fielding: Does it matter? URL: https://www.samford.edu/sports-analytics/fans/2024/Fielding-Does-It-Matter.

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, and Weinan E. 2021. Towards theoretically understanding why sgd generalizes better than adam in deep learning. URL: https://arxiv.org/abs/2010.05627, arXiv:2010.05627.

A Pitcher Statistics Used in the Model

In total, there are 61 pitcher stats for each season stats and the last 5 date stats

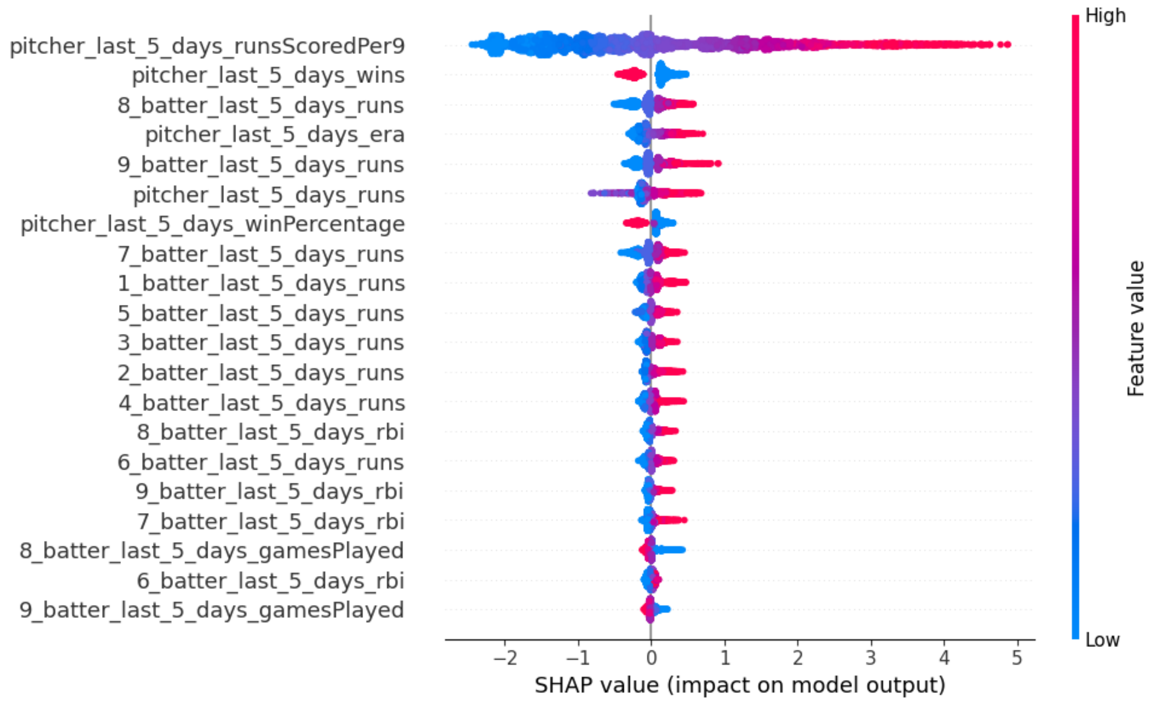
```
gamesPlayed, gamesStarted, flyOuts,
groundOuts, airOuts, runs,
doubles, triples, homeRuns,
strikeOuts, baseOnBalls,
intentionalWalks, hits,
hitByPitch, avg, atBats,
obp, slg, ops,
caughtStealing, stolenBases,
stolenBasePercentage, groundIntoDoublePlay,
numberOfPitches, era, inningsPitched,
wins, losses, saves,
saveOpportunities, holds,
blownSaves, earnedRuns,
whip, battersFaced, outs,
gamesPitched, completeGames, shutouts,
strikes, strikePercentage, hitBatsmen,
balks, wildPitches, pickoffs,
totalBases, groundOutsToAirouts,
winPercentage, pitchesPerInning,
gamesFinished, strikeoutWalkRatio,
strikeoutsPer9Inn, walksPer9Inn,
hitsPer9Inn, runsScoredPer9,
homeRunsPer9, inheritedRunners,
inheritedRunnersScored, catchersInterference,
sacBunts, sacFlies,
```

B Batter Statistics Used in the Model

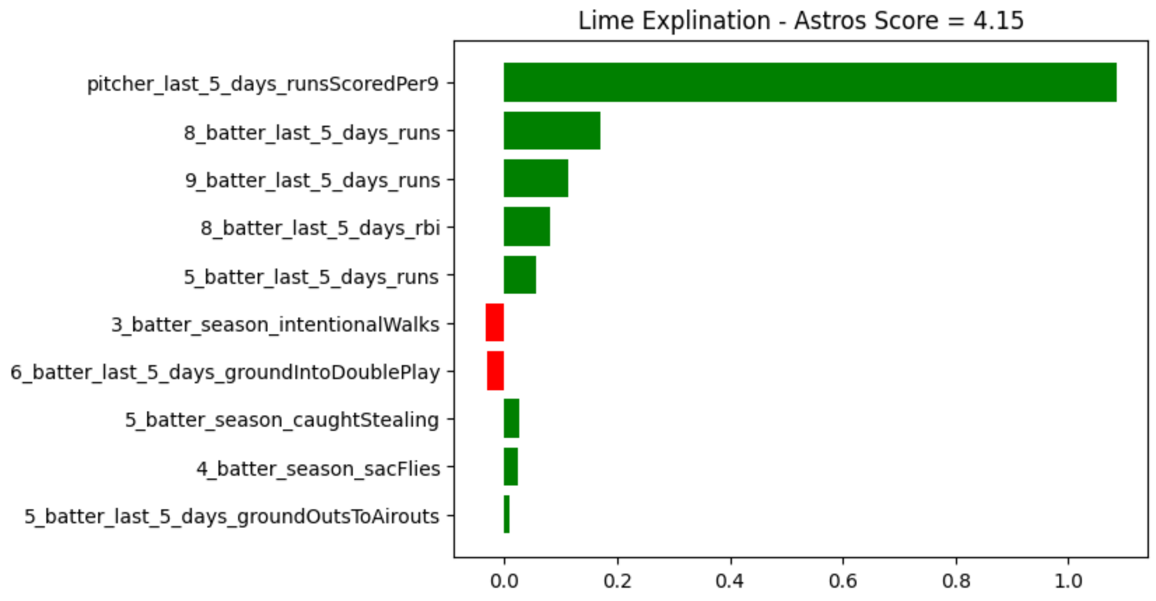
In total, there are 34 batter stats for each season stats and the last 5 date stats

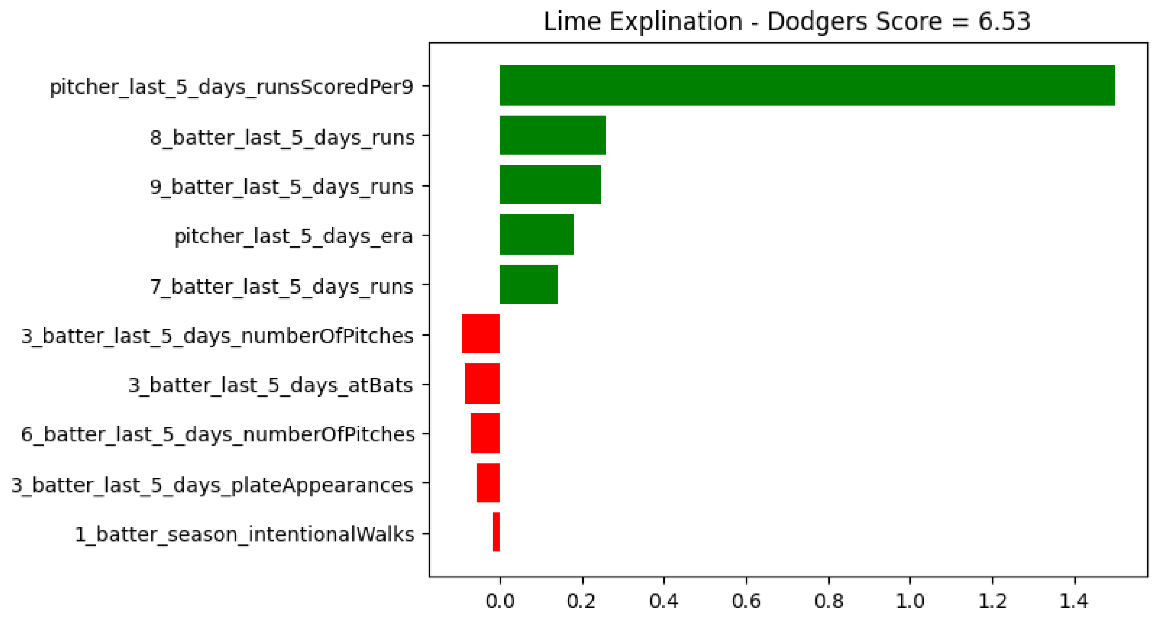
```
gamesPlayed, flyOuts, groundOuts,
airOuts, runs, doubles,
triples, homeRuns, strikeOuts,
baseOnBalls, intentionalWalks, hits,
hitByPitch, avg, atBats,
obp, slg, ops, caughtStealing,
stolenBases, stolenBasePercentage,
groundIntoDoublePlay, groundIntoTriplePlay,
numberOfPitches, plateAppearances, totalBases,
rbi, leftOnBase, sacBunts, sacFlies,
babip, groundOutsToAirouts,
catchersInterference, atBatsPerHomeRun
```

C SHAP Results



D LIME Results


Lime Explination - Astros Score = 4.15
pitcher_last_5_days_runsScoredPer9
8_batter_last_5_days_runs
9_batter_last_5_days_runs
8_batter_last_5_days_rbi
5_batter_last_5_days_runs
3_batter_season_intentionalWalks
6_batter_last_5_days_groundIntoDoublePlay
5_batter_season_caughtStealing
4_batter_season_sacFlies
5_batter_last_5_days_groundOutsToAirouts


Lime Explination - Dodgers Score = 6.53
pitcher_last_5_days_runsScoredPer9
8_batter_last_5_days_runs
9_batter_last_5_days_runs
pitcher_last_5_days_era
7_batter_last_5_days_runs
3_batter_last_5_days_numberOfPitches
3_batter_last_5_days_atBats
6_batter_last_5_days_numberOfPitches
3_batter_last_5_days_plateAppearances
1_batter_season_intentionalWalks