Building Machine Learning Models to Predict the Outcome of a Pitch in an MLB Game

Authors: Zachary Becker, Zachary Armand, Zhaofeng Li

Northeastern University

DS 5500 Capstone: Applications in Data Science

Spring 2025

March 18, 2025

Contents

Chapter 1

Abstract

How do you find the most optimal way for hitters and pitchers to approach an at-bat? What is the most likely outcome for a given pitch profile and batter? Understanding the most likely outcomes for each offering in a pitcher's repertoire helps a baseball team game plan and develop more in-depth approaches to each at-bat, thus providing a competitive advantage and putting teams in a better position to win games. This project utilizes machine learning techniques to predict expected wOBA given pitch data (XGBoost) or a sequence of pitch data (RNN). Models are evaluated using MSE and MAE for the regressions. We are currently seeing RMSEs around 0.35 for predicting expected wOBA, and are aiming to try and bring that closer to 0.3 in the next couple weeks. In baseball, it is common for "bad" pitches to produce positive results for the defense, and for "great" pitches to still be hit well, so there will always be predictions that are far off, raising our errors. Despite the inherent uncertainties, the models still provide valuable information about the likelihood of various outcomes, allowing teams to make better decisions regarding pitch selection, sequencing, and hitter approach in order to maximize their chance to succeed.

Chapter 2

Introduction and Related Work

2.1 Introduction

The main goal of the experimentation phase of our work is to attempt to create and tune the best performing models. Given our goal of predicting the most likely outcome of a pitch through expected wOBA, our experimentation will try to minimize the difference between our model outputs and the actual wOBA value for a given pitch or pitch sequence. For example, tuning the hyperparameters of our models or tweaking architecture will hopefully improve performance.

2.2 Related Work

Using machine learning and statistical methods to take advantage of the large amount of data associated with baseball to make useful predictions is hardly a new idea. Numerous other studies have explored leveraging MLB and Statcast data to make all sorts of predictions. However, most of these studies have either different targets and aims. For example, some studies will use baseball data to predict pitch type class[1] or the probability of a certain pitch outcome, like a strikeout.[2][8][9] Similar architectures have been created as well, such as linear neural networks to predict game matchup outcomes,[3] or LTSM recurrent neural networks and XGBoost models to predict pitch class type.[9] In one case, Statcast data and neural networks were combined with the aim of creating a model with the stated aim of improving real-time pitcher performance.[7]

The process of downloading data in bulk directly using pybaseball was influenced by a 2019 blog post. The author of this post used Statcast data to predict and correctly classify a batting matchup outcome (hit, out) using a variety of models, including k-Nearest Neighbors and XGBoost.[4]

While these previous works were interesting to read and informative as to how baseball data and machine learning have been combined, as far as we are aware, our work is one of, if not the only, to pair Statcast data with XGBoost and neural networks in order to predict wOBA. So, none of the previous works had a major role in shaping our work. Additionally, as a result, there were no other comparable baseline metrics that we could reference when evaluating our results. We therefore used common error metrics for regression-based tasks, such as root mean squared error and mean absolute error.

Chapter 3

Experimental Setup

Our work was used Python 3 exclusively for data retrieval, preprocesssing, and model creation and training. To retrieve Statcast data, we took advantage of the *pybaseball* python package,[5] which allows for seamless connection to the Statcast API to download large volumes of Statcast data. The XGBoost model used XGBoost, a common XGBoost library avaialble in multiple languages (`https://xgboost.readthedocs.io/en/stable/python/index.html`). The neural network models were implemented in PyTorch (`https://pytorch.org/`). Data preprocessing and model evaluation took advantage of common python packages like sci-kit-learn, numpy, pandas, and seaborne.

The neural network models were trained on two devices: an Apple M1 Pro with 16 GB of memory, taking advantage of the device's integrated GPU and PyTorch's MPS backend (`https://pytorch.org/docs/stable/notes/mps.html`), and a NVIDIA GeForce GTX 1060 with 3 GB of memory.

Chapter 4

Dataset and Preprocessing

4.1 Dataset

We exclusively used data from Major League Baseball's Baseball Savant,[6] a platform that provides access to MLB's Statcast data. Statcast captures detailed measurements of every pitch thrown in MLB games, including pitch velocity, spin rate, release point, and batted-ball metrics (exit velocity, launch angle, etc.).

Statcast systems are integrated into every MLB stadium. Data collection for Statcast began being phased in around 2015, making it a relatively new but rich source of baseball analytics data. Statcast fully replaced the previously widespread PITCHf/x data collection system in the 2017 season.

For our work, we are using Statcast data from the 2024 MLB season. This data is publically available for download. To access the data, we used the python package *pybaseball* to directly access the Baseball Savant API via Python.

The dataset that we used has a row for each pitch thrown in the 2024 MLB season, and contains a wide selection of variables that describe player and game contexts, pitch data, batted ball data, and more. There were around 750,000 pitches thrown in the 2024 MLB season, but we only looked at the pitches that were put into play. Some of the key features used to do the pitch profile segmentation include the pitch type, the velocity, spin rate, spin axis, vertical break, horizontal break, release extension, and release height. For the hitter-specific data, we look at the most common launch angles and exit velocities by pitch location (divided into 16 zones) as well as the by pitch type. Some hitters will fare better against specific types of pitches and specific locations of the strike zone, so we of course need to take that into account. We predicted the exit velocity and launch angle of a batted ball, which are then used in calculating the expected weighted on-base average (wOBA) of the play.

4.2 Preprocessing Steps:

To preprocess the data, we only used regular season games, removed any at-bats with a score difference greater than 6 and removed any pitches with less than 3 appearances in the 2024 season. Any rows with missing values in the pitch type, release speed, spin axis, release extension and effective speed columns were dropped.

For the neural network models, special care was taken to ensure that groups of at-bats were preserved when preparing data. This was done because the neural network models expected sequential pitching data, instead of one-off observations. Groups that did not have a value for *hit_into_play'* or an estimated wOBA in any of their rows (pitches) were removed. Additionally, any at-bats with more than 9 pitches total were removed in order to reduce the dimensionality of the final dataset while keeping 99% of the total data. At-bat sequences were then padded with 0 values if they contained fewer than 9 pitches, in order to ensure consistent dimensionality of the input data.

Chapter 5

Training and Validation Process

5.1 Training Setup:

Hyperparameters in both the XGBoost and RNN models were tuned using a grid search. The XGBoost uses 3 stratified folds for cross validation, tuning the max depth, learning rate, number of estimators, subsample, sample of features used for each tree, as well as the L1 (alpha) and L2 (lambda) penalties.

Regularization techniques included dropout, Adam optimizer L2 weight decay, and layer normalization were used, with the amount of dropout and weight decay chosen during hyperparameter tuning.

The linear neural network model tuned batch size, optimizer learning rate, dropout percentage, and Adam optimizer L2 weight decay penalty. The RNN model tuned batch size, optimizer learning rate, dropout percentage, Adam optimizer L2 weight decay penalty, and number of GRU layers to use. When training the neural network models, a batch size of 32 was used with 200 training epochs for both models. The Adam optimizer was used, with learning rate and weight decay values being tuned during hyperparameter tuning. A GRU recurrent cell was chosen for the RNN model.

5.2 Final Testing:

We do an 80/20 train-test split for the XGBoost, using random state 4400 for the XGBoost. The neural networks included an additional validation set for model evaluation, with a 60/15/15 train-test-validation split using the random split function in PyTorch.

Chapter 6

Results

Our main metric for evaluating our models is root mean squared error (RMSE). The results for our models when evaluated against the testing data set can be seen in Table 6.1. Currently, the models are all performing with a RMSE of around 0.36, with the Linear Neural Network having the lowest score by a small value.

Given the original target wOBA scores have a range of $\{0.0, 2.0\}$, these RMSE values represent a generally well-performing loss for such a complicated predictive task. They represent a reasonable prediction of the outcome of an at-bat.

It was originally unexpected that the RNN model did not perform as well as the simple linear model when tested against the same testing set. Our hope was that the recurrent model would be able to sequentially process pitch data and capture the dependencies as an at-bat progressed. The difference in testing and validation data, as well as final test RSME, of the the linear and RNN models can be seen in Figure 6.1. The gap between the testing and validation values of the RNN might suggest that the model wasn't able to generalize as well, and overfit to the training data. Further investigation into why this gap occurred, and further strategies to improve performance will be tested.

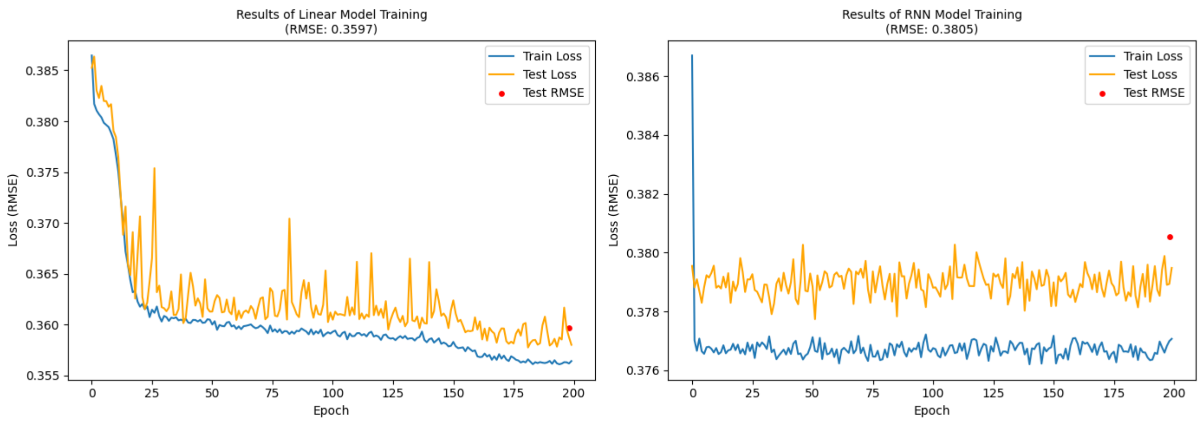

Figure 6.1: Linear and RNN Model Performance

In the future, we would want to further reduce the error of our models in order to achieve better predictive power. We will investigate different ways to achieve this reduction in our final iterations.

Table 6.1: Model Performance on Testing Set

Model	**RSME**
XGBoost	0.3627
Linear Neural Network	0.3597
RNN	0.3805

Chapter 7

Evaluation Metrics and Validation

To evaluate our models, we use Mean Squared Error and Mean Absolute Error for our main regression performance metrics, and include accuracy for parts of the neural networks. MSE is typically the go-to performance metrics for regression tasks like this one, but we also wanted to include MAE to treat errors equally. As mentioned earlier, it is common for "bad" pitches to produce positive results for the defense, and for "great" pitches to still be hit well, so we don't want to skew the MSE too much in the case that a hanging breaking ball down the middle leads to a weakly hit ground ball, or a terrific slider on the outside corner is hit for a home run.

For the Recurrent Neural Networks, we make sure to maintain the order of pitches during training, which prevents data leakage and ensures that we're mimicking in-game scenarios. We monitor MSE and MAE over our iterations, refining hyperparameters based on our experimentation, which allows us to tweak the model to best predict expected wOBA.

Chapter 8

Discussion and Iterative Improvements

In this phase, we were able to further tune the hyperparameters using Grid Search for the XGBoost, and testing out different architectures for the neural networks. Trying out more RNN layers, attention mechanisms, and weight initializations all helped to slightly improve the model's performance.

For our hitter-specific features, we had the plan of including prior launch angle and exit velocity combinations for each pitch type in each location zone, but for many of the offspeed pitches, there weren't enough data points to get meaningful results. There are plenty of data points for fastballs in the middle of the strike zone, for example, but for a pitch such as a high and outside splitter, there may only be one or two data points since there aren't many pitchers that throw a splitter, and only a small fraction of splitters are thrown high in the zone. We shifted to incorporating prior success based on pitch type and zone separately to combat this.

To build off our current findings, we can add more years of data, and test on the 2025 season. It would also likely be beneficial to incorporate external data about weather, ballpark dimensions and altitude, or try and predict other batted ball metrics. There are several other ways to gauge quality of contact made, such as the barrels per batted ball event percentage, or hard hit percentage. We also are aiming to create more visualizations to aid in our storytelling, especially for an audience that does not have the same technical background or knowledge about the game of baseball.

Chapter 9

Conclusion

This phase allowed for the improvement in both the development and evaluation of our models. We are able to get a better sense of the results given by the models, and get an improved estimate of how good of a result we can truly achieve given the stochastic nature of an at-bat. Going forward, we can try to incorporate more years of data, and begin to test on the 2025 season, which has just started. There is an opportunity to incorporate external data about weather, ballpark dimensions, altitude, etc., as well as predict other kinds of batted ball metrics, such as the number of barrels per batted ball event, or hard hit percentage.

Chapter 10

Reproducibility

The GitHub link for our project is: `https://github.com/zybecker/DS-5500`. The repository contains the requirements and any information needed to clone the repo.

The Statcast data can be accessed at Baseball Savant: `https://baseballsavant.mlb.com/statcast_search`. An example notebook of how to download 2024 Statcast data using pybaseball is included in our GitHub repository.

Bibliography

[1] Adam Attarian et al. "A Comparison of Feature Selection and Classification Algorithms in Identifying Baseball Pitches". In: (2013).

[2] Jasmine Barbee. "Prediction of Final Pitch Outcome in MLB Games Using Statistic Learning Methods". MA thesis. California State University, Long Beach. Department of Mathematics and Statistics, 2020. URL: https://link.ezproxy.neu.edu/login?url=https://www.proquest.com/dissertations-theses/prediction-final-pitch-outcome-mlb-games-using/docview/2492950717/se-2?accountid=12826.

[3] Mei-Ling Huang and Yun-Zhi Li. "Use of Machine Learning and Deep Learning to Predict the Outcomes of Major League Baseball Matches". In: *MDPI* 11.10 (2021), p. 4499. URL: https://www.mdpi.com/2076-3417/11/10/4499.

[4] N. Koenig. *Predicting Performance Using Baseball Data.* AIO: Bridging the Gap Between Data Science and IO. 2019. URL: https://nkoenig06.github.io/performance-baseball.html.

[5] James LeDoux. *pybaseball.* GitHub repository, available at GitHub. 2023. URL: https://github.com/jldbc/pybaseball.

[6] MLB Advanced Media. *Baseball Savant.* 2025. URL: https://baseballsavant.mlb.com/.

[7] Stephen Eugene Otremba Jr. "SmartPitch: Applied Machine Learning for Professional Baseball Pitching Strategy". MA thesis. Massachusetts Institute of Technology. Department of Electrical Engineering and Computer Science, 2022. URL: https://hdl.handle.net/1721.1/145144.

[8] Yifan PI. *Predict Next Baseball Pitch Type with RNN.* [Semester project, Stanford University]. 2018. URL: https://cs230.stanford.edu/projects_spring_2018/reports/8290890.pdf.

[9] C.-C. Yu, C.-C. Chang, and H.-Y. Cheng. "Decide the next pitch: A pitch prediction model using attention-based LSTM". In: *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW).* 2022, pp. 1–4. DOI: 10.1109/ICMEW56448.2022.9859411. URL: https://doi.org/10.1109/ICMEW56448.2022.9859411.