Introduction

As a data scientist, a significant portion of my time on this project was dedicated to refining and cleaning the data, conducting exploratory data analysis (EDA), and performing feature engineering. I drew upon techniques and approaches that I had previously used in projects with similar objectives. My initial step was to create a plan for transforming the raw pitch data obtained from the file into a format that could be inputted into machine learning models. This involved identifying and addressing missing values, inconsistencies, and outliers, as well as selecting and engineering features that were relevant and informative for predicting the target variable. Through iterative cycles of data cleaning, EDA, and feature engineering, I was able to develop a high-quality dataset that was ready for modeling.

Data Cleaning

I began by performing general data cleaning, where I addressed minor issues that surfaced during the data wrangling and feature engineering stages. These included identifying and removing columns that would not hold significant relevance to the objective, and replacing unknown pitch types with NaN values.

Game state features(Markov Chain Analysis)

Next, I proceeded to create game state features. By utilizing the balls and strikes data, I generated a feature that represented the current count for each at-bat, which I categorized into three groups based on whether it favored the pitcher, the batter, or if it was neutral (pc_-1.0,pc_0.0,pc_1.0). Pitchers will alter their approach depending on the count, making this information valuable. I also created a feature to represent the score differential(run_diff) and modified the baserunner features to a binary format, indicating whether or not a runner was on each base. The three variables on_1b, on_2b and on_3b have either a 0 or a 1 for if a baserunner is on that base. These features could potentially reveal insights into pitcher tendencies, as runners on base may impact the pitch selection.

Pitch categorization features

Unfortunately, a minute fraction of the pitches in the data lacked pitch type classification, which may have been caused by system malfunctions or other factors. To maintain the sequence of pitches on a game-by-game basis, rather than discarding these rows from the dataframe, I saved the discarding until after the data was fully aggregated. I categorized the pitches into two categories: offspeed and fastball. I dropped all the pitches that were unique to the point where they didn’t even show up on the bar chart. The two categories included the specific pitches: three general categories: fastballs (such as 4-seam, 2-seam, cutter, and sinker), non-fastballs (curveballs, sliders, knuckle curves, change-up, knuckle ball).

Game flow features

Although pitchers may have a tendency to throw pitches of a certain type at a certain frequency based on historical data, there is a considerable amount of variability on a game-by-game basis for various reasons. One significant contributing factor is that each pitcher may have more or less command and control over a certain pitch on a given day. For instance, weather conditions

like precipitation or humidity may impact their grip and ability to spin a curveball, which may cause them to rely more heavily on other pitches like fastballs and change-ups. Alternatively, they may struggle to throw strikes or get batters out with their fastball, but have more success with their slider or curveball on a particular day. To try and capture some of these variables, I developed some features based on the pitcher's recent pitch history.

To capture the data for the last three pitches, new columns were added for the velocity and pitch type. The features were one hot encoded and were labeled as followed: (start_speed_lag1, start_speed_lag3, start_speed_lag2, pitch_group_lag1_Fastball, pitch_group_lag1_Non-Fastball, pitch_group_lag2_Fastball, pitch_group_lag2_Non-Fastball, pitch_group_lag3_Fastball, pitch_group_lag3_Non-Fastball)

Rest of the features

Here is the final list of all features and the weightings for the importance with random forest:
1. pc_-1.0 (0.143377)
2. pcount_at_bat (0.124149)
3. pc_1.0 (0.104475)
4. pcount_pitcher (0.098512)
5. at_bat_num (0.050169)
6. start_speed_lag1 (0.047968)
7. start_speed_lag3 (0.047307)
8. pc_0.0 (0.047026)
9. start_speed_lag2 (0.044302)
10. pitch_group_lag1_Fastball (0.038253)
11. pitch_group_lag1_Non-Fastball (0.036537)
12. inning (0.027602)
13. pitch_group_lag3_Non-Fastball (0.024217)
14. on_2b (0.023794)
15. pitch_group_lag3_Fastball (0.021051)
16. outs (0.018763)
17. pitch_group_lag2_Non-Fastball (0.017224)
18. pitch_group_lag2_Fastball (0.015860)
19. is_final_pitch (0.015064)
20. run_diff (0.014554)
21. b_height_inches (0.013860)
22. on_3b (0.013718)
23. on_1b (0.007482)
24. stand (0.002475)
25. top (0.002260)

I converted the height of batter to inches to try and encapsulate differing strike zones. Stand is whether the batter bats righty or left, 1 for righty and 0 for lefty. The pitch_group_lag features are just the OHE 1 for fastball and 0 for non-fastball of the last 3 pitches. The rest of the features were original variables or defined earlier.

Modeling

Once I had collected the data, I subjected it to various models to identify the most effective one and determine the best parameters to use. However, due to the limited time available, I was unable to accomplish some tasks, such as carrying out model parameter optimization through GridSearch, and applying more complex cost functions for better model evaluation. It's important to note that fitting the models can be a time-consuming process, and the computation cost was a big limiting factor for me. The models I ran included Logistic Regression, Neural Network, and Random Forest. SVM took too long to run so it was omitted.

I ultimately decided to go with the Logistic regression since it had the lowest computational expense and all of the models were very similar in performance. I computed a confusion matrix as well as other important evaluating metrics like accuracy, F1, precision and the feature importance rankings for the RF.

The classification report shows the precision, recall, and f1-score for each class, as well as the overall accuracy of the model. In this case, the model is predicting the "fastball" class with an accuracy of 0.63, which means it correctly predicted the class for 63% of the test set samples. The precision of the model for the "fastball" class is 0.65, which means that when it predicted the "fastball" class, it was correct 65% of the time. The recall for the "fastball" class is 0.93, which means that the model correctly identified 93% of the actual "fastball" samples.

The model is predicting the "non-fastball" class with an accuracy of 0.63, which means it correctly predicted the class for 63% of the test set samples. The precision of the model for the "non-fastball" class is 0.52, which means that when it predicted the "non-fastball" class, it was correct 52% of the time. The recall for the "non-fastball" class is 0.13, which means that the model correctly identified only 13% of the actual "non-fastball" samples.

In general, an accuracy of 0.63 is decent but not great. The precision and recall for the "fastball" class are both good, but the precision for the "non-fastball" class is low, indicating that the model may be biased towards predicting the "fastball" class. This could be due to class imbalance or other factors, and it may be worth exploring ways to improve the performance of the model on the "non-fastball" class, such as adjusting the class weights or using a different algorithm.

Future Improvements

Computational expense

Moving forward, there are several areas in which I can improve this model. Currently, computational efficiency is still a big limiting factor. Even retraining the model with about 75% of one year's data can take a significant amount of time on a modern laptop. To address this I could utilize parallel processing, which can speed up the computational time for models. This can be done using parallel computing frameworks like Apache Spark, Dask, or Ray. I could also utilize dimensionality reduction techniques like Principal Component Analysis (PCA) which can

help to reduce the number of features in a dataset by finding a smaller set of features that capture most of the information in the original dataset.

Feature Engineering
Enhanced feature engineering is the main improvement. Engineering features that will accurately delineate the batter/pitcher relationship would be a huge upgrade. Using the description of the result of the preceding pitch and the x, y location variables, to encapsulate the exact result would be monumental. Understanding if a batter hit a sharp line drive that happened to get caught or if a batter chased the pitch would be beneficial. Also better understanding the batter pitcher relationship, for instance, if a batter like Rafael Devers has a history with a pitcher like Garrett Cole and Garret tends to pitch him in specific spots and specific pitches, would be a great feature to implement. Finally, it would be advantageous to try to capture any tendencies that the pitcher may exhibit when throwing pitches after specific events, such as after giving up a walk, a base hit, run, or home run, or after striking out a hitter. These tendencies could be subconscious or unknown to the pitcher, and uncovering them could provide valuable insights.

OHE Features
One potential improvement for categorical features that have been one-hot encoded is to implement an ordinal or sliding scale. This method could be especially useful for features like the current count of the batter or the description of the pitch beforehand. Using an ordinal or sliding scale could more accurately capture the relationship between categories, which could enhance the performance of models that rely on distance or similarity calculations between instances. Another advantage of this approach is that it could help to reduce the dimensionality of the feature space, which can be advantageous when there are many categorical features. However, it's worth noting that this approach may necessitate additional computational resources for both training and evaluating the model.

Pitcher specific
Targeting specific pitchers. Predicting pitches on the league as a whole is seemingly difficult and through my research and reading of other papers, targeting specific pitchers seems more pragmatic.

Conclusion
Ultimately, the key to improving the accuracy of the model lies in feature engineering. While the current set of features has enabled the model to predict fastball or non-fastball with a reasonable degree of accuracy, there are likely many additional features that could be engineered or implemented to better predict the specific pitch type. Incorporating more relevant features could help the model to make more nuanced predictions, allowing for more precise identification of the next pitch type.

Appendix
I originally attempted to predict the specific pitch type, but due to time constraints, I found that the model was not efficient enough. I then divided the pitches into three types which yielded

decent results. However, in order to simplify the model and improve its performance, I ultimately decided to reduce the pitch types to just two.