Capstone Project 1 - Major League Baseball Fastball Prediction Model

Final Report

by Greg Jacobs (jacobs.greg@gmail.com)

I. Problem Statement

Strategy in Major League Baseball is ever-involving. With hitters increasingly focused on launch angles and elevating the ball, it is unsurprising that teams are becoming more reliant on home runs to score. In fact, in 2019 the teams collectively hit an eye-popping 6,776 home runs, which is roughly a 10% increase over the previous record of 6,105 home runs hit in 2017. (Data compiled and published by Baseball Almanac at https://www.baseball-almanac.com/hitting/hihr6.shtml). This trend is very likely to continue in the coming seasons.

It also is generally-accepted that fastballs are particularly ripe for batters to hit for home runs given their increased velocity and lack of movement relative to other pitch types. Indeed, Fantasy Labs, a website focused on providing strategic advice to fantasy sports players, noted that fastballs were the most common pitch type to be hit for home runs. (https://www.fantasylabs.com/articles/home-run-trends-part-3-pitch-type/). And simple common sense dictates that a batter will have an even greater chance of hitting a home if he knows that the pitcher is going to throw a fastball - one only needs to watch batting practice or the home run derby to see this confirmed.

The goal of this project is to build a model that attempts to accurately predict when a pitcher will throw a fastball. The targeted clients for this model are professional, semi-professional, and collegiate baseball teams, including the players, managers, and other personnel. The organizations could use this data to inform their batters to anticipate fastballs under certain circumstances, which in turn should lead to the team hitting more home runs. Conversely, organizations could use this information to inform their pitchers as to when fastballs are most often thrown and coach the pitchers to throw a different pitch under those circumstances. Professional sports gamblers could also use the model to inform their wagers. These are just a handful of what is likely many uses for an accurate fastball prediction model.

II. The Datasets

This project uses the MLB Pitch Data 2015-2018 dataset that is publicly-available on Kaggle at (https://www.kaggle.com/pschale/mlb-pitch-data-20152018). I have chosen to build my model using two .csv files from that dataset. The first file, pitches.csv, charts various data for each pitch thrown during each of the four seasons from 2015 through 2018. The second file, atbats.csv, contains various static data for each at-bat from each of the four seasons from 2015 through 2018. The two files have the following dimensions:

pitches.csv == 2,870,000 x 40; and

atbats.csv == 740,000 by 11.

One of the categorical variables in the pitches file, pitch_type, classifies the pitch type for each pitch thrown. The variable contains over fifteen distinct labels, including FF for a four-seam fastball, FT for a two-seam fastball, and FC for a cut-fastball. The dataset also has a label for split-fingered fastballs, but we will treat that category as a non-fastball for the purpose of this project because the pitch behaves and is used more similarly to other breaking balls (such as a sinker). I will be able to use these labels to create a new binary variable to classify each of these three types of fastballs as a fastballs and the remaining categories of pitches as non-fastballs, which will become the target of my analysis.

III. Exploratory Data Analysis

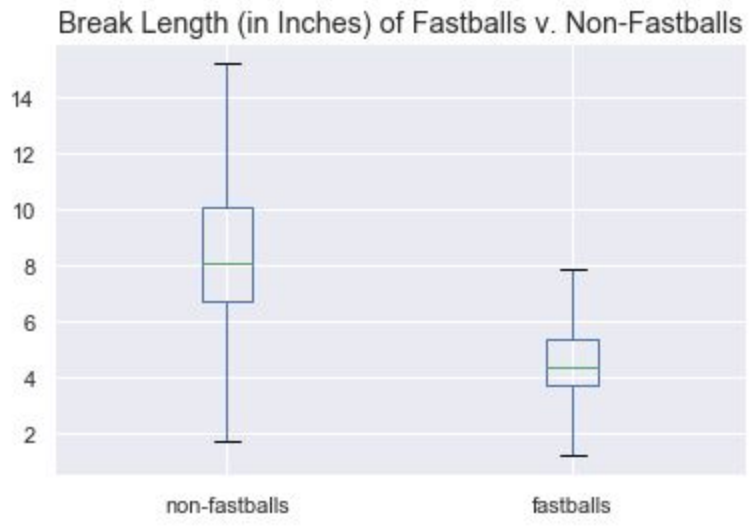
Once merged and cleaned,[1] the dataset has two types of features to explore.

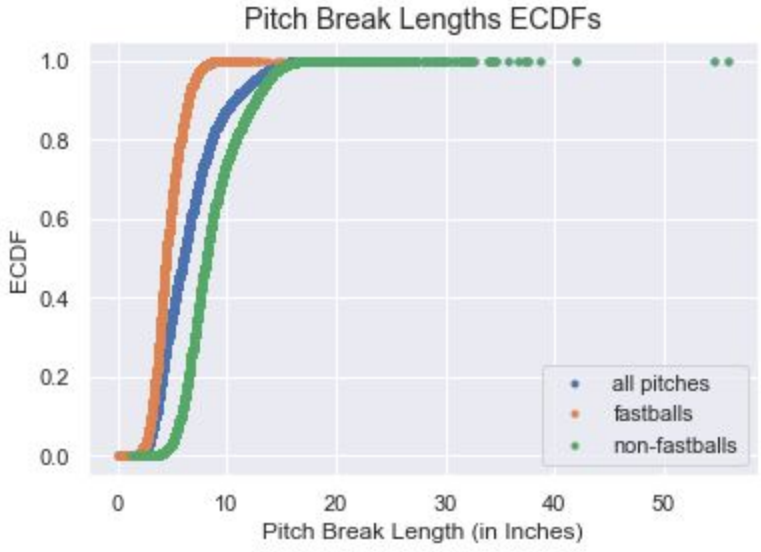
A. Post-Release Continuous Variables

The first type is a number of continuous variables documenting characteristics of each pitch once the pitcher has released it, including each pitch's speed, location, spin/rotation, and movement. Not surprisingly, each one of these features are statistically significant to determining whether the pitch is a fastball. Here is an example

[1] For a detailed explanation of the steps taken to wrangle and clean the data, please see the Fastball_Prediction_Model_Final_Notebook file included in the GitHub repository for this project, which you can access at https://github.com/gmj110680/Springboard/tree/master/Capstone_Project_1.

from the continuous variable ‘break_length’, which measures the amount of break the pitch had (in inches):





As the above graphs demonstrate, fastballs have significantly less break on average than non-fastballs. Exploratory analysis resulted in similar results for the other seven post-pitcher release continuous variables.

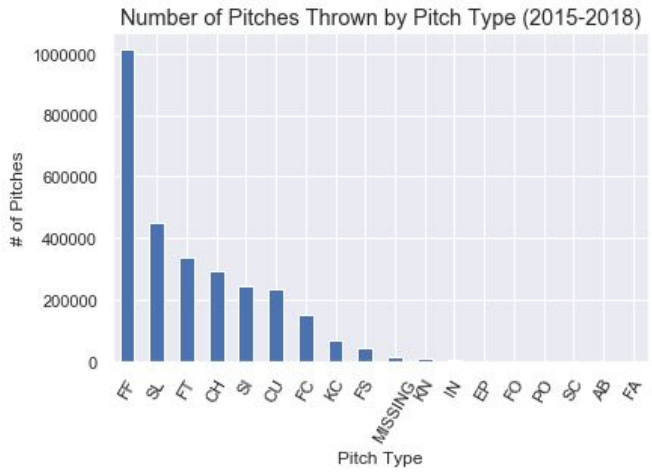
The issue, of course, is that all of this data is largely unhelpful for the direct purpose of my model given that we do not receive this information until after the pitch has been thrown. In other words, it is too late to be helpful for the current pitch. However, it is still

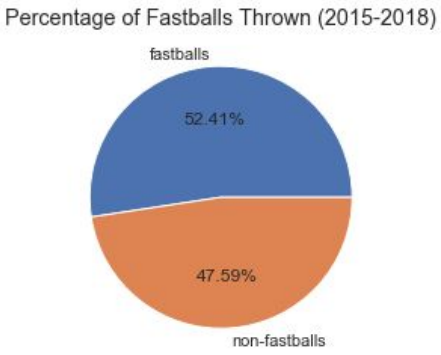
valuable to know that we almost instantly can classify a pitch as a certain type once it is thrown. We may be able to use this data to tell our model information about the previous pitch thrown to see if that information is predictive of the next pitch.

B. Pre-Pitch Categorical Variables

The second type of features in the dataset are pre-pitch categorical variables. These are more important for the building of my model because, unlike the post-release continuous variables discussed above, they are known prior to the pitch being thrown. Some examples of these circumstances are runners on base, the batting count, the game score, the pitch number in the at-bat, the handedness of the pitcher and batter, and the number of outs.

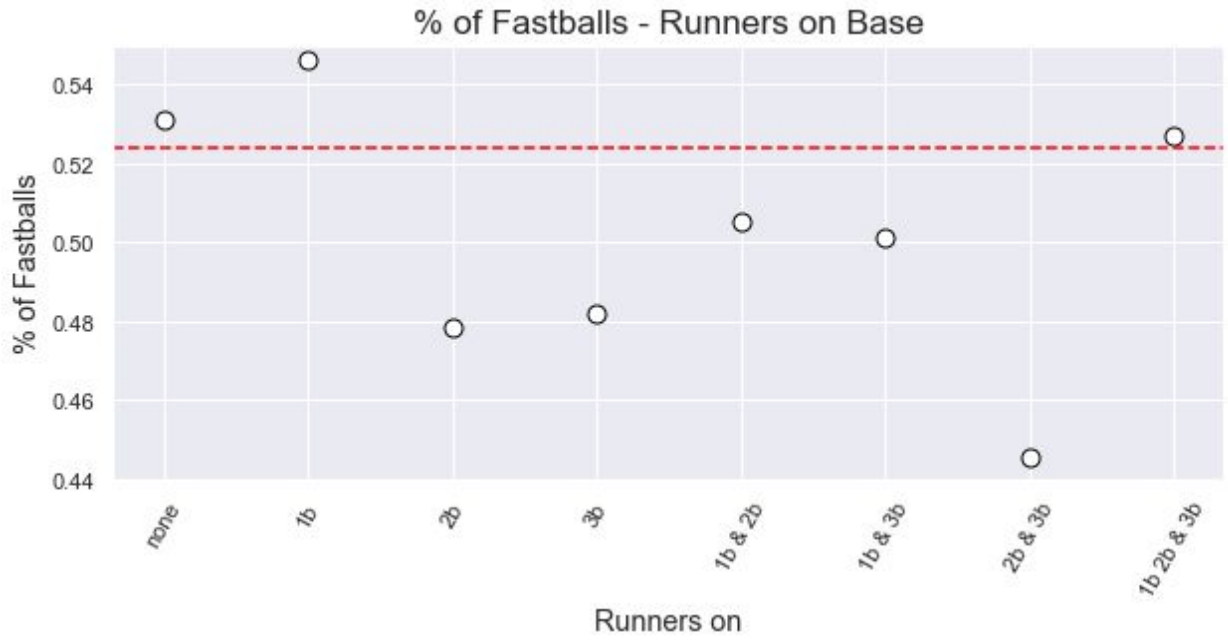
In order to accurately evaluate the various categorical variables, I first needed to identify a default baseline percentage regarding how often a pitcher throws a fastball. This can be done by calculating the percentage of pitches that are labeled as a fastball in the entire dataset:





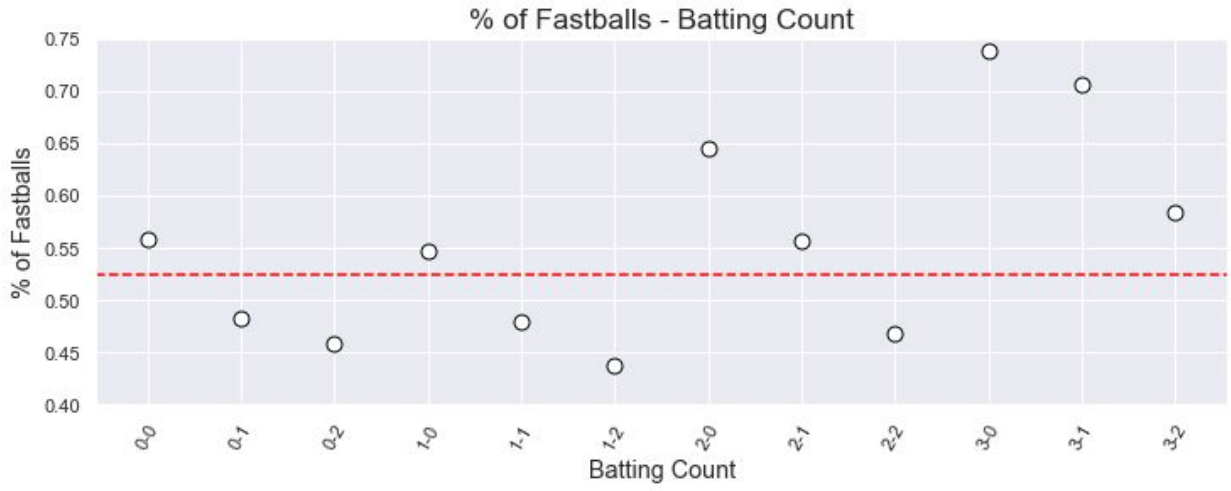
As we can see, a pitcher threw a four-seam, two-seam, or cut-fastball approximately 52.41% of the time. I can use this number as a baseline when analyzing the various categorical data variables. A variable is not overly significant to the extent that it correlates to a fastball usage at or around that same 52.41% frequency - the goal is to identify the circumstances in which that frequency increases (or decreases) materially from that baseline number.

Runners on Base



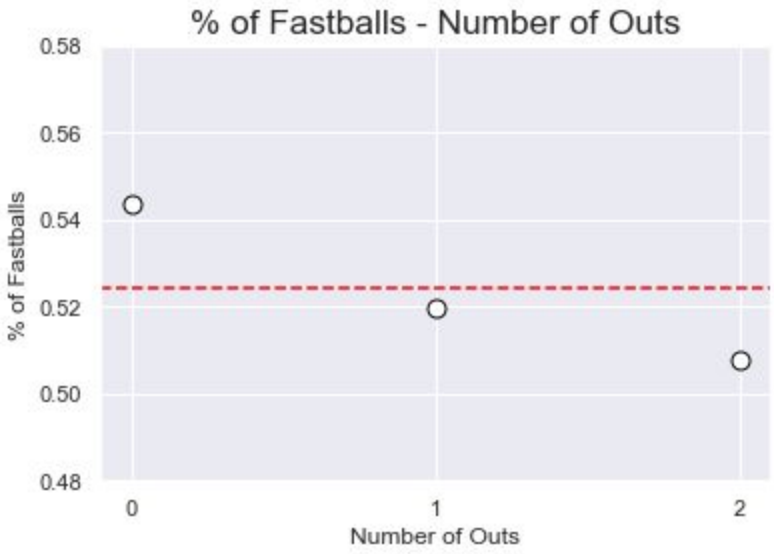
The most significant influence on fastball usage is whether there is at least one runner in scoring position (on second or third base). Interestingly, however, the fastball usage spikes back up to above average when the bases are loaded, likely because a walk in those circumstances results in a run scored.

Batting Count



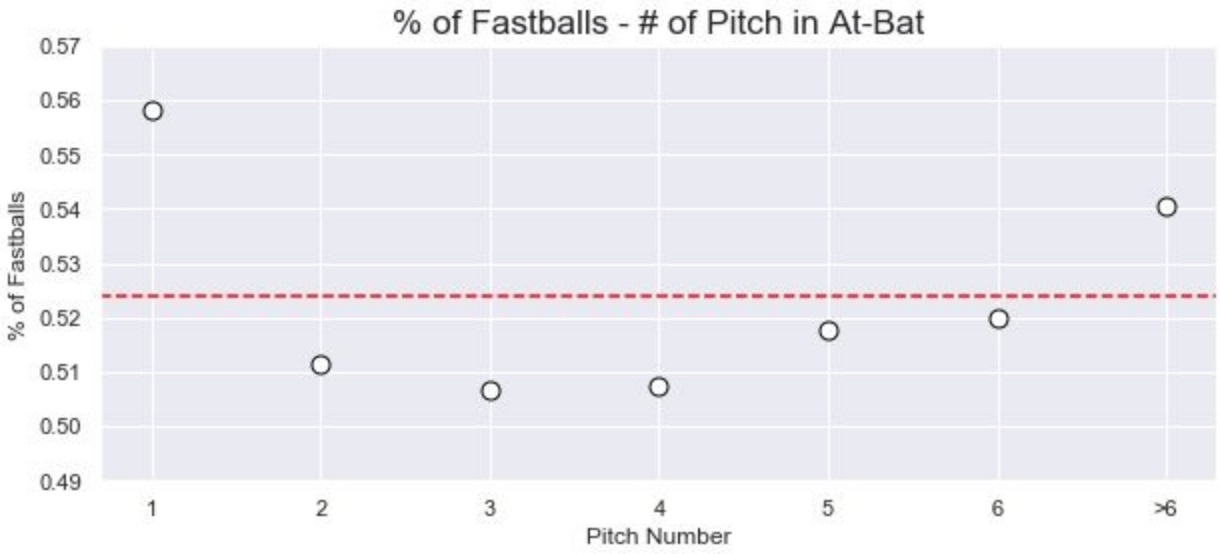
A pitcher's fastball usage decreases when he is ahead in the count and increases when he is behind in the count (likely because he needs to try to avoid walking the batter).

Number of Outs



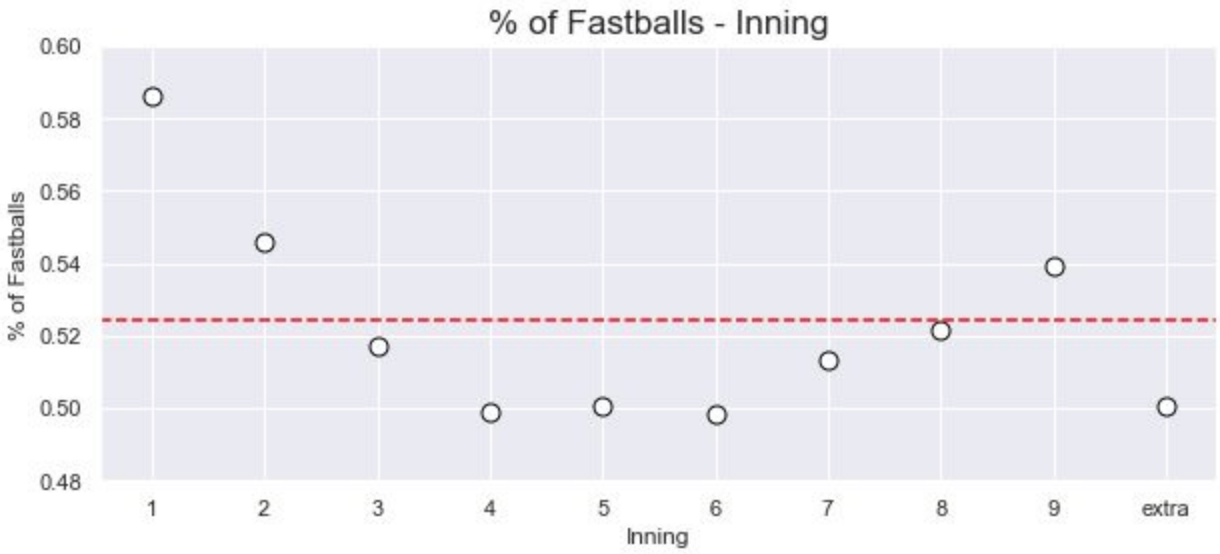
Fastball usage decreases depending upon the number of outs.

Pitch Sequence



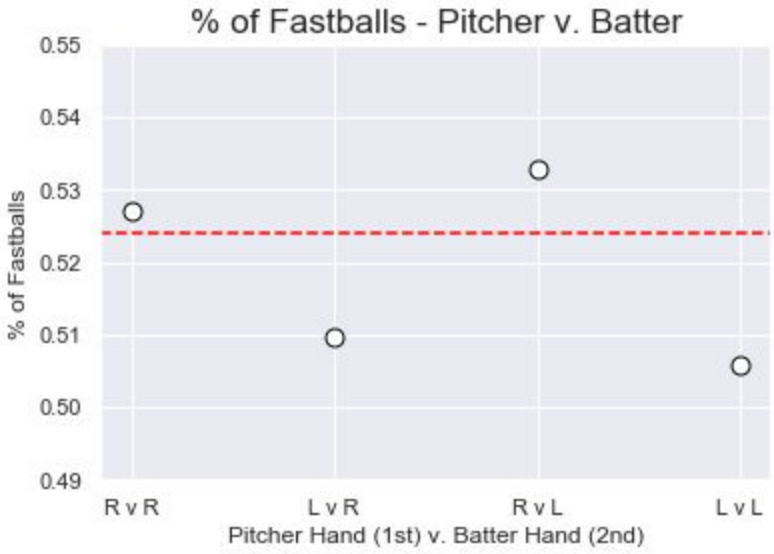
Pitchers throw more than average amounts of fastballs by a considerable amount on the first pitch of an at-bat and any pitch after the sixth pitch in a long at-bat.

Inning



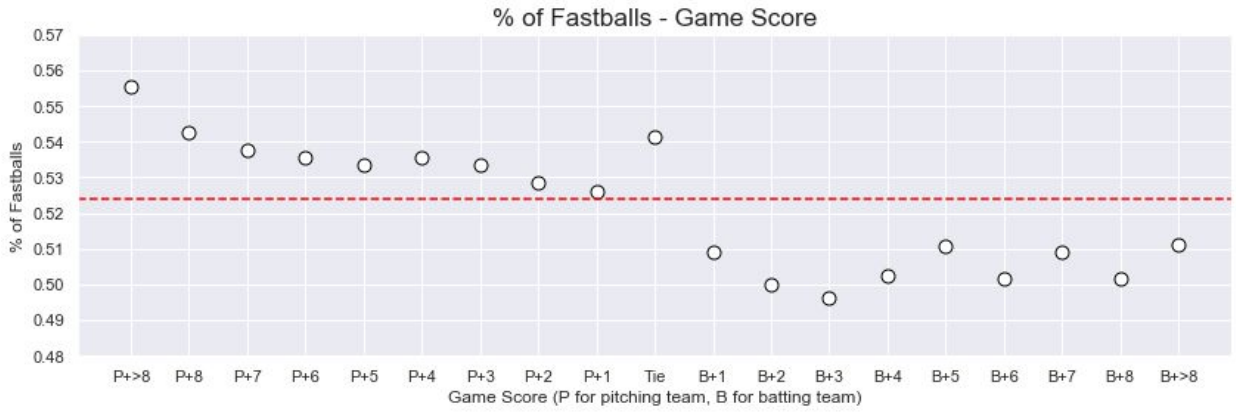
Fastballs are thrown higher than average in the early innings and the ninth inning. The middle innings appear to be when the least amount of fastballs are thrown.

Pitcher/Batter Dominant Hand



Right-handed pitchers throw fastballs more often than left-handed pitchers regardless of the handedness of the batter.

Game Score



Pitchers throw more fastballs than usual when their team is winning or the game is tied and significantly less fastballs when their team is behind.

In summary, it certainly appears as though some correlation exists between certain game circumstances and the likelihood that a pitcher will throw a fastball on the next pitch. I therefore am hopeful that a model can predict when a fastball will be thrown at a rate higher than the default 52.41 percent.

IV. Initial Machine Learning Models Test Run

At this point in the project, I decided to feed the dataset into three “out-of-the-box” classifiers to see how they would perform. Because my dataset has over 2.8 million samples and would be a significant time investment to train the models on the full dataset, I elected to feed the classifiers approximately 5% of the data (145,000 pitches) and confirmed that the sample data had the same 52/48% split between fastballs and non-fastballs.

I also used only the pre-pitch categorical features discussed in the previous section. Including the post-release continuous features would no doubt dramatically improve model performance, but would defeat the purpose of this exercise by using information not available to the batter prior to the pitch being thrown.

The initial “out-of-the-box” models yielded the following accuracy on the test set from the sample data:

- Logistic Regression Classifier: **0.551117**;
- Stochastic Gradient Descent Classifier: **0.543834**; and
- Random Forest Classifier: **0.526731**.

The results were not encouraging. The “out-of-the-box” classifiers performed only marginally better than the default 52.41% accuracy that you could achieve simply by guessing fastball everytime. So, it appeared at least on the surface that Major League Baseball pitchers were fairly adept at mixing up their pitch selection in the various game circumstances that I fed into my models.

I had not, however, fed any information into the models regarding the previous pitch thrown by the pitcher. That information would be known by the batter and was available in my dataset, so I decided to add some additional features to the dataset fed into my models.

V. Additional Feature Engineering

I was fairly confident that information about the previous pitch in a pitch sequence should improve the accuracy of my models. The major obstacle to this approach, however, was that building previous pitch features would involve having to iterate over the samples to include the data only when relevant (i.e., only to pitches that are not the first pitch of a particular at-bat), which was very time-consuming and computationally expensive given that there are over 2.8 million pitches in my dataset. I ultimately decided to add the following features:

- **Inning Pitch Count** - this feature provides how many pitches a pitcher has thrown in a particular inning. I could only tally this efficiently on an inning-by-inning basis (rather than a pitch count for the entire game) given how the data was aggregated in my dataset;

- **Previous Pitch Type** - this feature provides the pitch type of the immediate previous pitch thrown to the batter using the pitch type labels (Once this data was compiled, I converted it to categorical features using dummy variables); and

- **Vertical/Horizontal Previous Pitch Location** - these two features provide the vertical and horizontal pitch location of the immediate previous pitch to a batter in

a particular at-bat. These were built off of the ‘px’ and ‘pz’ continuous variables discussed above.

I ended up settling on using only the previous pitch type and location features due to how computationally expensive it was to build these features over a 2.8 million sample dataset. Had I not been under such a time constraint, I would have built a previous pitch feature for each of the continuous variables discussed in the exploratory data analysis section of this report. I ultimately chose to simply use the previous pitch type label knowing that the particular label somewhat aggregates that data into one feature.

VI. Second Machine Learning Models Test Run

Having added the previous pitch features to the dataset, I fed the updated 5% sample data into some “out-of-the-box” models to see if I had improved performance. Here are the resulting test set accuracy numbers for each model:

- Logistic Regression Classifier: **0.601517**;
- Stochastic Gradient Descent Classifier: **0.591917**;
- Random Forest Classifier: **0.569545**; and
- Gradient Boosting Classifier: **0.601848**.

As we can see, adding features regarding the previous pitch in a pitch sequence significantly improved the “out-of-the-box” models. Given that the Logistic Regression and Gradient Boosting Classifiers were able to produce an accuracy of over 0.6, I elected to use those two models going forward.

VII. Hyperparameter Tuning

For the Logistic Regression Classifier, I tuned the ‘C’ regularization hyperparameter as well as the solver for optimal performance on the same 5% sample data, which produced the following results on the training and test sets of the data:

- C: 1000;
- Solver: ‘lbgfs’;
- Training Set Accuracy: **0.6036**; and
- Test Set Accuracy: **0.6017**.

For the Gradient Boosting Classifier, I tuned the ‘learning_rate’, ‘n_estimators’, ‘max_depth’, ‘min_samples_split’, ‘min_samples_leaf’, ‘max_features’, and

‘sub_sample’ hyperparameters for optimal performance on the same 5% sample data, which produced the following results:

- ‘learning_rate’: 0.010;
- ‘n_estimators’: 900;
- ‘max_depth’: 9;
- ‘min_samples_split’: 600;
- ‘min_samples_leaf’: 50;
- ‘max_features’: 9;
- ‘subsample’: 0.70;
- Training Set Accuracy: **0.6251**; and
- Test Set Accuracy: **0.60750**.

As demonstrated by the numbers, the hyperparameter tuning process did not significantly improve either model. The Logistic Regression Classifier improved by a negligible amount while the Gradient Boosting Classifier improved by approximately 0.5%. This likely means that it will be difficult to achieve an accuracy percentage significantly better than these values based on the features currently available to the models even when using the complete dataset (assuming the sample size is a comparable distribution to the complete dataset).

VIII. Final Models Performance

Overall

Using the optimal hyperparameter values calculated during the tuning process, I fed the entire 2.8 million sample dataset into the two fully-tuned models to get the following final results:

Metric	***LRC***	***GBC***
Training Set Accuracy	0.602586	0.611946
Test Set Accuracy	0.601775	0.608487
Non-fastballs Precision	0.63	0.61
Non-fastballs Recall	0.40	0.48
Non-fastballs F1-score	0.49	0.54
Fastballs Precision	0.59	0.61
Fastballs Recall	0.78	0.73
Fastballs F1-score	0.67	0.66
ROC area under curve	0.651080	0.661980

The Gradient Boosting Classifier generated an overall accuracy rate of 0.6085, almost a full percentage point better than the Logistic Regression Classifier's rate of 0.6018. This spread was similarly reflected in the models' respective ROC curve AUC score, as the Gradient Boosting Classifier generated a score of 0.6620 compared to the Logisitic Regression Classifier's 0.6510. Therefore, the Gradient Boosting Classifier should be preferred if overall accuracy is a team's priority.

The decision may not be so straightforward, however, if a team prioritizes identifying one type of pitch over another. Let's examine the accuracy percentages for each pitch type to provide a clearer picture on model performance.

Non-Fastballs

Pitch Type	***LRC***	***GBC***
Slider	0.38	0.46
Changeup	0.35	0.43
Sinker	0.55	0.60
Curveball	0.33	0.42
Knuckle Curve	0.34	0.42
Splitter	0.39	0.47
Missing	0.70	0.66
Knuckleball	0.65	0.69
Intentional Ball	0.85	0.93
Eephus	0.60	0.66
Pitchout	0.33	0.46
Screwball	0.66	0.71

As we can see from the above table, the Gradient Boosting Classifier identified each type of non-fastball at a higher accuracy rate than the Logistic Regression Classifier (with the exception of the pitches that were missing a pitch type classification). The Gradient Boosting Classifier accordingly should be used by any team that prioritizes identifying when the next pitch will be a non-fastball.

It is worth noting that neither classifier does particularly well in identifying non-fastballs. The Gradient Boosting Classifier correctly classified 47% of non-fastballs, which significantly outperformed the Logistic Regression Classifier's recall rate of 40%. However, the Logistic Regression Classifier generated a slightly higher precision rate (0.63 v. 0.62), largely because it classifies so few pitches as non-fastballs.

Fastballs

Pitch Type	*LRC*	*GBC*
Four-seam Fastball	0.77	0.72
Two-seam Fastball	0.81	0.76
Cut Fastball	0.78	0.73

As demonstrated above, the Logistic Regression Classifier was more accurate in identifying each of the three types of fastballs by a significant margin. The Logistic Regression Classifier correctly classified 78% of fastballs, which significantly outperformed the Gradient Boosting Classifier's recall rate of 73%.

However, the Logistic Regression Classifier generated only a marginally higher F1-score with respect to fastballs than the Gradient Boosting Classifier (0.67 v. 0.66), and the Gradient Boosting Classifier produced a slightly higher precision rate (0.60 v. 0.59). This can be accounted for by the fact that the Logistic Regression Classifier predicts fastballs at a much higher overall rate than the Gradient Boosting Classifier - the Logistic Regression Classifier predicted approximately 70% of the pitches to be fastballs, significantly higher than the Gradient Boosting Classifier's rate of approximately 63% of the pitches. The Logistic Regression Classifier therefore produces more Type I errors/False Positive predictions than the Gradient Boosting Classifier (203,569 v. 178,111), but less Type II errors/False Positive predictions (81,874 v. 102,521).

All that being said, a team should consider using the Logistic Regression Classifier if its top priority is to maximize the number of fastballs identified correctly.

Model Features

The following tables provide the 15 model features that produced the highest accuracy (on the left) and the 15 model features that produced the lowest accuracy (on the right):

Feature	***LRC***	***GBC***
prev_pitch_IN	1.00	1.00
px_prev_(3.467, 4.0)	0.95	0.94
px_prev_(-4.008, -3.467)	0.94	0.94
prev_pitch_KN	0.90	0.90
prev_pitch_SI	0.88	0.88
b_count_3	0.72	0.73
px_prev_(-3.467, -2.933)	0.73	0.71
prev_pitch_EP	0.67	0.71
px_prev_(2.933, 3.467)	0.74	0.71
prev_pitch_SC	0.96	0.67
prev_pitch_FC	0.66	0.66
px_prev_(-1.867, -1.333)	0.64	0.65
pz_prev_(3.4, 3.8)	0.63	0.64
px_prev_(3.8, 4.2)	0.63	0.64
inning_9	0.64	0.64

Feature	***LRC***	***GBC***
prev_pitch_CU	0.54	0.55
prev_pitch_CH	0.55	0.55
prev_pitch_KC	0.55	0.56
prev_pitch_FO	0.57	0.56
prev_pitch_FS	0.56	0.57
prev_pitch_SL	0.56	0.57
inning_4	0.57	0.58
pz_prev_(-1.006, -0.6)	0.59	0.58
prev_pitch_PO	0.61	0.58
pz_prev_(-0.6, -0.2)	0.58	0.58
inning_3	0.58	0.59
inning_5	0.58	0.59
inning_6	0.58	0.59
pz_prev_(-0.2, 0.2)	0.59	0.59
s_count_2	0.58	0.59

As the above tables demonstrate, with very few exceptions the two models produced roughly the same accuracy for each of the model features. On average, the Gradient Boosting Classifier tends to perform slightly better on most features, accounting for its superior overall accuracy discussed above.

The features that the models produce the highest accuracy mostly comprise of certain previous pitch type and location data. Interestingly, however, certain other previous pitch types and location data produced the least accurate results. The models also produced low accuracy numbers on pitches in the middle innings.

Overall, this demonstrates that the previous pitch features that I added to the model produced the most skewed results (which I knew implicitly given how the models' overall performance numbers improved significantly once these features were added). Future work on these models should include focusing on these features to see if any more specific patterns/insights can be identified.

I also would like to take a deeper dive into the pitches that occur in the middle innings. While I am skeptical that it is the middle inning features themselves that are causing the poor accuracy results, it could be indicative of other patterns/trends in the data that occur more often in the middle innings, which thereby cause the models to mislabel more pitches.

Logistic Regression Coefficients

Next, I will examine the largest positive and negative coefficient values that the logistic regression classifier assigned to the features. The largest positive values indicate the features that will most significantly influence a positive (fastball) classification, and the largest negative values indicate that features that will most significantly influence a negative (non-fastball) classification. Here are the results:

Feature	***Coefficient***
prev_pitch_FC	3.039
prev_pitch_FT	2.808
prev_pitch_FF	2.743
prev_pitch_FO	2.500
prev_pitch_PO	2.438
prev_pitch_CU	2.389
prev_pitch_CH	2.296
prev_pitch_KC	2.292
prev_pitch_SL	2.232
prev_pitch_FS	2.183
prev_pitch_EP	1.916
b_count_3	0.725
pz_prev_(-1.006, -0.6)	0.355
pz_prev_(-0.6, -0.2)	0.260

Feature	***Coefficient***
prev_pitch_IN	-6.606
pitch_num_2	-2.442
pitch_num_9	-2.365
pitch_num_10	-2.348
pitch_num_8	-2.308
pitch_num_7	-2.278
pitch_num_6	-2.261
pitch_num_3	-2.186
pitch_num_5	-2.124
pitch_num_4	-2.080
prev_pitch_SC	-1.037
s_count_2	-0.937
s_count_1	-0.601
s_count_2	0.580

The results suggest that the logistic regression classifier prioritizes virtually every previous pitch type as an important indicator that the next pitch will be a fastball, with 12 of the 15 being different previous pitch types. Interestingly, the top 3 values (FC, FT, FF) are the three types of fastballs that comprise our fastball label. This is an area that will require future investigation - if certain of these pitch types don't correlate with the next pitch being a fastball, it may make sense to remove these features from the model to see if it would improve the model's accuracy.

It also determined that a batting count with 3 balls is relatively significant to the next pitch being a fastball, which is consistent with what we observed during our exploratory data analysis. Finally, it found slight significance in the previous pitch having a negative vertical location (i.e., hitting the dirt below the plate). Those likely are quite a small sample size and therefore probably not significantly impacting the model's overall performance.

Conversely, the classifier assigns large negative value coefficients to features it has learned correlate to non-fastballs. The outlier negative feature is for the previous pitch being an intentional ball. This makes logical sense because intentional balls are thrown when a pitcher intentionally walks a batter, in which case the pitcher throws four consecutive intentional balls. The other previous pitch type listed is a screwball, which is also an outlier given how few pitchers throw screwballs.

The most significant trend that the classifier learned is that pitch numbers 2-10 of a batting count are indicative of non-fastballs. This is somewhat consistent with what we discovered during exploratory data analysis, though we observed that the frequency of fastballs increased to above average from pitches 6-10.

The classifier also assigned significance to batting counts with one or two strikes. This is largely consistent with what we observed during exploratory data analysis.

Gradient Boosting Classifier Feature Importances

Unlike the logistic regression classifier, the gradient boosting classifier only produces positive “feature importance” values to each model feature on a normalized (0-1) scale. These values will indicate to us which features are most important to the model in making its key decisions within the decision trees, with a higher value indicating that the feature is more influential. Here are the 15 most (on the left) and least (on the right) features in the model:

Feature	***Importance***
prev_pitch_SI	0.379
prev_pitch_FF	0.081
b_count_3	0.057
s_count_2	0.054
prev_pitch_FT	0.044
prev_pitch_FC	0.038
inning_pitch_count	0.030
s_count_1	0.022
b_count_1	0.021
prev_pitch_KN	0.020
on_2b	0.016
pitch_num_2	0.016
prev_pitch_SL	0.015
b_count_2	0.014

Feature	***Importance***
prev_pitch_SC	0.000000
prev_pitch_PO	0.000003
prev_pitch_FO	0.000017
px_prev_(-3.467, -2.933)	0.000018
prev_pitch_EP	0.000086
pitch_num_10	0.000115
px_prev_(2.933, 3.467)	0.000131
pitcher_lead_-9	0.000145
px_prev_(-2.933, -2.4)	0.000150
pitcher_lead_-8	0.000271
px_prev_(2.4, 2.933)	0.000273
pitch_num_9	0.000290
pitcher_lead_9	0.000291
pz_prev_(-0.6, -0,2)	0.000291

Here, we can see that the Gradient Boosting Classifier has determined the previous pitch slider feature as the most important feature by a significant margin. It also has assigned relative importance to the three types of fastballs that make up our `fastball` label, which is consistent with what the Logistic Regression Classifier observed (indicative that the next pitch will also be a fastball). In future work, I intend to investigate pitches following sliders to determine if the classifier is correct in assigning such a high level of importance to that feature.

The model has also assigned relatively high importance to batting counts with 3 balls or 2 strikes. This is consistent with what we observed during exploratory data analysis (3-ball counts are indicative of fastballs, 2-strike counts of non-fastballs). It is less clear why it assigned relative importance to 1-ball, 1-strike, and 2-ball counts. It may be worth experimenting in removing these features to see if that would improve the model's overall performance.

The model also observed importance in the number of pitches thrown by the pitcher in the inning (it could be that more fastballs are thrown later in the inning to avoid walks) and runners on second base (indicative of non-fastballs). It also applied some importance to the second pitch of an at-bat, which we observed during exploratory data analysis to be more likely to be a non-fastball.

In analyzing the features that the Gradient Boosting Classifier assigned the least amount of importance, it becomes clear that most if not all of the features tend to be outlier values. Among the features are previous pitch types that are rarely thrown (screwballs, pitchouts, eephus), game scores in which the pitcher's team is winning (or losing) by a wide margin, extreme previous pitch location values, and the last pitches of abnormally long at-bats (9 or greater). It may make sense to consider dropping these outlier features in future iterations of this model, though I would not expect that to significantly impact the model's overall performance given the relative infrequency of these features in the dataset and the low importance the model assigns to them.

IX. Summary/Conclusions

This project demonstrates that it is possible to analyze pre-pitch circumstances to predict whether a pitcher is going to throw a fastball as the next pitch at a higher accuracy rate than the default frequency percentage of 52.41%. It is important to note that the models struggled to outperform the default rate without information regarding the previous pitch despite observing correlations between fastball frequency and certain of these circumstances (runners on base, batting count, pitch sequence, game score, etc.). It is interesting to note that, while correlations between certain game circumstances and fastball frequency do exist, the same overall game circumstances did not significantly improve the algorithms' ability to predict fastballs when considering all of that data in the aggregate.

Once we included information about the pitcher's inning pitch count, the immediate previous pitch type thrown, and the location of the immediate previous pitch, the models'

respective accuracy improved significantly by approximately 5-8%. Teams therefore should be able to use one of the final models to correctly identify when a pitcher will throw a fastball or non-fastball more often than competitors not using such a model.

The Gradient Boosting Classifier should be used for most circumstances given its overall superior performance. As explained in detail above, the Gradient Boosting Classifier performed substantially better in identifying non-fastballs and generated higher accuracy numbers on most of the model features. It also appears to have more potential to improve through future work.

That being said, a user should consider using the Logistic Regression Classifier under circumstances in which the user is concerned only (or primarily) with identifying when a fastball will be thrown and is less concerned about mis-identifying non-fastballs as fastballs.

X. Future Work for Potential Model Improvements

I believe that I potentially could improve the performance of the final models generated by this project with some additional work. The main hurdles that I encountered with this project were the large size the data (over 2.8 million pitches) and the limited computational power and time available. With more time and access to more computational power, I think the models could potentially be improved through exploration of one or more of the following:

- **Adding more previous pitch data as additional features**. I was only able to add certain previous pitch features to the dataset given how much time/computational power it took to build them. If I had more time and computational power, I would suggest adding the other previous pitch data available for the immediate previous pitch (pitch speed, break length, spin rate, etc.). I also would suggest adding the same information for each previous pitch for an entire at-bat (rather than only the data from the immediate previous pitch). I believe this has the potential to improve the model performance given how significantly the previous pitch features improved my model in this project;
- **Adding interaction terms using `PolynomialFeatures`**. My exploratory data analysis demonstrated some correlations between certain pre-pitch game circumstances and a pitcher's fastball usage (e.g., runners in scoring position, whether the pitcher is ahead or behind in the count, whether the pitcher's team is winning or losing, etc.). However, the models were unable to capture much (if

any) significance of these circumstances when making fastball predictions in the aggregate. Adding interaction terms would potentially allow the models to more accurately identify the significance of two (or more) of these circumstances being present at the same time. I was unable to incorporate this step given my time and computing power limitations, but in future work I would like to explore whether interaction terms would improve the models;

- **Removing certain existing features**. As explained above, I would like to do some further feature-specific analysis/testing to determine whether the models may perform better without inclusion of certain features. In particular, the models appeared to be quite sensitive to the previous pitch features that I added. By comparing the feature coefficient/importance values against one another and to the observations made during my exploratory data analysis, I think I may be able to identify some features that are not improving model performance. Removing some features would reduce the size of the dataset and make it more manageable to analyze; and
- **Tuning the hyperparameters with a larger portion of the data.** I only used approximately 5% of the data to tune my model hyperparameters given my time and computational power constraints. Tuning the models on the smaller data sample did not meaningfully improve their performance. Tuning the models on a greater percentage of the data potentially could produce different optimal hyperparameter values for the models, though not likely unless the 5% sample of my data happened to be a poor representative of the entire dataset. I would like to perform this exercise to confirm that I identified the optimal hyperparameters for my models.