Predicting and Recognizing Pitches Using Pre-Event and Post-Event Data

Paul Lawrence
Department of Combinatorics and Optimization
University of Waterloo
p4lawren@uwaterloo.ca

1 Introduction

To say that baseball and advanced statistical analysis have a close, at times symbiotic relationship, would be an understatement. Due to detailed historical records, large numbers of discrete events per game, and our increasing ability to measure and quantify various qualitative aspects of the actions the players make, baseball serves as an ideal playground for those wishing to experiment with data-related optimization techniques. The analytic revolution of the 2000's in baseball was detailed and brought to public knowledge with Michael Lewis' *Moneyball* in 2003 - however, with even more advanced mathematical and statistical knowledge, as well as newer technologies such as Rapsodo, Hawkeye, Statcast, and PITCHf/x, modern baseball is optimized to a far greater degree than ever before. [1]

One particular area of interest is in pitch prediction. Common knowledge dictates that if a batter knows what pitch is coming, they will be more successful at hitting it (although there has been some recent pushback to this at the major league level [2]). For evidence of the power of pitch prediction, we need look no further than the Houston Astros, who infamously orchestrated a technology-aided sign-stealing scandal during parts of 2017 [3]. In short, if we can build models that more accurately predict the next pitch, we can gain an advantage over the competition.

Unsurprisingly, there are several examples of machine learning being used in pitch prediction, both in the academic [4][5] and blog [6][7] settings. However, the majority of these do not predict pitch classes based on each individual class, but rather a more binary distinction of "fastball" or "not fastball". Originally, during brainstorming and proposing for this project, I intended to produce a study that improved upon these results, by including not only the current game state but also by including information on previous pitch data. However, during research and testing, all possible avenues for this kind of analysis eventually lead to using timeseries data, which after numerous attempts to implement (and advice from the professor) proved too copmlicated for me. Thus, I have settled on simply exploring the class of models for pitch prediction, and attempting to reproduce some of the results mentioned in the references, while expanding the scope beyond simple binary classification to a more robust multi-classification model. To add variety to the project, instead of simple pitch prediction, we attempt two types of classifying problems: prediction and recognition.

The first type of classification is self-explanatory: we input a game state and attempt to predict which type of pitch will be thrown next. Due to the game-theoretic nature of baseball, accuracy is less important here than getting an accurate probabilistic spread, so we pay more attention to the negative log loss than we do to the accuracy metric.

The second type of classification is pitch recognition. More specifically, given a large amount of data on a pitch that was recently thrown, can we correctly classify the type of pitch? Applications are numerous, most notably Major League Baseball's own pitch classification method that is used to identify pitches during live broadcasts. In some sense, we are attempting to reproduce MLB's classification algorithm using their own data, since the data that we use is pulled from MLB's pitch

database itself.

This project attempts both forms of prediction to varying degrees of accuracy. Inspired by [4] and [6], we use several different types of models, and analyze their performance on the datasets of different pitchers. Some methods, such as Support Vector Machines on mixed post-and-pre-event data for pitch prediction, are highly unsuccessful, and some other methods are more successful. Surprisingly, there seems to be little public published work on thie classification problem.

2 Dataset

The two main cruxes of this project were data collection/cleaning and modeling. For data collection, I used Pybaseball [8], a python module created specifically for baseball data collection in python. Pybaseball has many built-in functions to scrape various websites that contain baseball data, but my main target was Baseball Savant, which contains most of the relevant data I intend to use such as pitch velocity, spin rates, location, and batter tendencies.

Our dataset is composed of statcast data pulled from Baseball Savant for each individual pitcher (Justin Verlander, Sandy Alcantara, Aaron Nola, Shane Bieber, and Adam Wainwright). For every pitcher, we scrape every pitch they have thrown between the 2018 and 2022 seasons, which works out to be between about 10,000 and 13,000 pitches, depending on the pitcher. Each pitcher has a specific repertoire they throw. For example, Verlander throws exclusively curveballs, changeups, four-seam fastballs, and sliders, while Wainwright throws fastballs, sinkers, curveballs, changeups, and sliders. For Verlander, a slice of the dataset looks like the following:

pitch type	game date	release speed	pos x	pos z	at bat number	pitch number	balls	strikes
FF	2022/10/04	93.9	−1.36	7.1	38	4	1	2
SL	2022/10/04	87.3	−1.55	6.98	38	3	1	1
SL	2022/10/04	86.2	−1.65	6.97	38	2	1	0
SL	2022/10/04	85.4	−1.5	6.99	38	1	0	0
SL	2022/10/04	84.7	−1.46	7.14	37	5	1	2
SL	2022/10/04	88.1	−1.5	7.01	37	4	1	2
CU	2022/10/04	78.9	−1.62	7.02	37	3	1	1
SL	2022/10/04	86.8	−1.76	7.01	37	2	1	0
SL	2022/10/04	87.2	−1.54	7.04	37	1	0	0
FF	2022/10/04	95.8	−1.35	7.09	36	7	3	2

The dataset can be split into what we call **pre-event** and **post-event** data. Pre-event data is all of the game state data that we have access to before the pitch is thrown. This includes handedness of the pitcher, handedness of the batter, the count, year, date, inning, number of pitches thrown in the game so far, and game score. Post-event data is all of the pitch data that we have access to. This includes velocity, spin rate and axis, location in the strike zone, etc. We use several different combinations of data depending on the model, which we will get into later.

3 Methods

We attempt to analyze our statcast data using several different methods. The first method tests specifically for pitch prediction, using a mixture of pre-event and post-event data. To normalize data and remove outliers, we train and test specifically for six different classes of pitches - three types of fastballs (Four-seamer, Cutter, Sinker), two types of breaking balls (Slider and Curveball) and one type of offspeed (Changeup). In this sense we improve upon previous models as we attempt a multi-class classification model rather than a simple "fastball or not fastball" binary classification model. The pre-event data is kept mostly unmodified, except for removing a few outliers (for example, Zack Grienke threw a total of 5 cutters between 2018 and 2022). Post-event data includes a large amount of pitch metrics, including zone, location, velocity, and spin rates, while removing all batted-ball data except those that are available for every pitch (for example, the result of the pitch, as in, "batted ball", "swinging strike", "ball", etc). During training, we utilize both pre-event and post-event data, but during testing only input pre-event data. To mitigate overfitting, we randomly apply dropout with probability $\frac{1}{5}$ to all post-event data during training. We train each

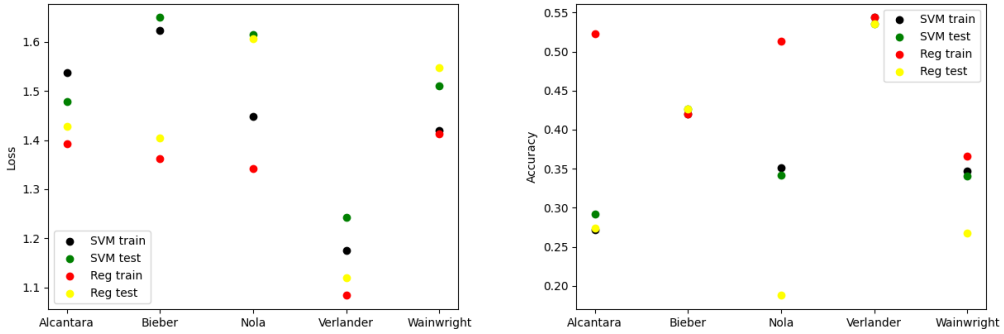
pitcher-specific model (since pitch types and data are hyper-specific to each pitcher) using 80% of the data and test on the remaining 20%.

The second experiment tests pitch prediction using only pre-event data. Here, we sacrifice more data (we really only have a few relevant data points of pre-pitch data, and we only have 14 points of data at all) for less chance of overfitting - this is essentially the model in [7]. The third experiment tests pitch recognition using only post-event data, again taking away all batted-ball data. This way, we only use data collected specifically from the action of the pitch, and not any data related to the batter or outcome of the pitch. Lastly, we run a final experiment where we concatenate all the pitchers' data files and learn with a pitcher-agnostic approach, sacrificing specificity for more data.

Inspired by [4], we attempt two different classification models. We use python's Sklearn module to implement a multiclass Support Vector Machine (SVM) and a Linear Regression model. Respectively, this tests our data using the one-versus-one and one-versus-all methods. As it turns out, the latter is highly more effective. All datasets and implementation details can be found on my github repository [9].

4 Results

Training and testing on the first model (a mix of pre-event and post-event data, testing for pitch *prediction*) we can see that Regression is far superior to SVM in both minimizing training loss and maximizing accuracy. We have the following results on our train and test sets:



While an accuracy above 33% is good for events that are so game-theoretic, it seems like the model is simply just predicting the same pitch every single time. For Sandy Alcantara, the per-pitch accuracy (in percentage points) is:

Model	Fastball	Sinker	Curveball	Slider	Changeup
Test Regression	0	83.24	0	0	29.96
Train Regression	0	0	0	0	100
Test SVM	0	0	0	0	100
Train SVM	0	0	0	0	100

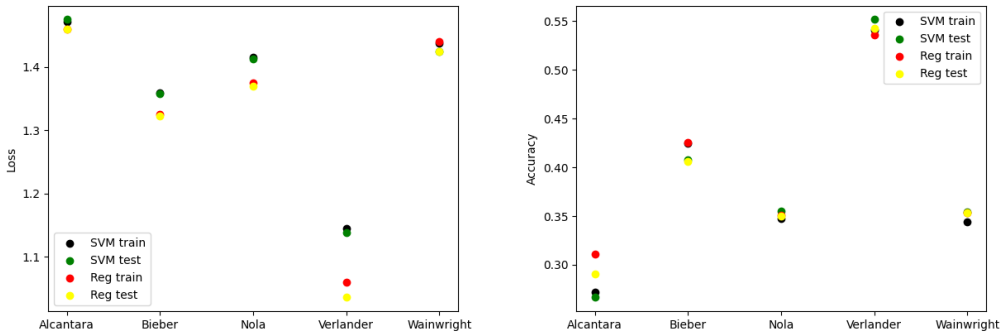
This suggests that, for the most part, the models almost always predict "Changeup", and furthermore that the prediction vector we get is not changing much from pitch to pitch. To illustrate this, we can look at the prediction vector for several test data points, first for regression and next for SVM:

True Label	FF%	SI%	CU%	SL%	CH%
FF	0.195	0.419	0.004	0.027	0.354
SL	0.203	0.418	0.004	0.028	0.347
SL	0.198	0.419	0.004	0.028	0.351
SI	0.198	0.418	0.004	0.028	0.351
FF	0.202	0.418	0.004	0.028	0.347

True Label	FF%	SI%	CU%	SL%	CH%
FF	0.346	0.220	0.005	0.083	0.347
SL	0.346	0.220	0.005	0.083	0.347
SL	0.346	0.2204	0.005	0.083	0.347
SI	0.346	0.220	0.005	0.083	0.347
FF	0.346	0.220	0.005	0.083	0.347

The left table has more variance, but the interesting point to make here is that the right table (corresponding to our Support Vector Machine) assigns probabilities almost identical

for every single pitch, suggesting that it has a very difficult time using the data we've provided it to say anything substantial. Moving toward the second model, we modify the dataset such that we *only* look at pre-pitch data. The initial results are below:



As we can see, both models perform slightly better, and compare similarly across pitchers. For accuracy, regression improves more than SVM, which barely improves. We also see that the model is not interpreting pitches as uniformly as before. Again, looking at Alcantara's per-pitch accuracy, SVM still has the same per-pitch issues as the previous dataset, but Linear Regression has dramatically improved:

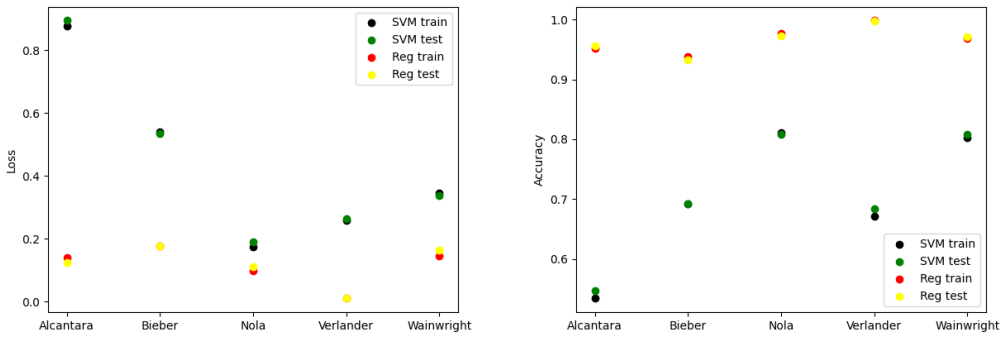
Model	Fastball	Sinker	Curveball	Slider	Changeup
Test Regression	53.66	42.85	0	0	25.56
Train Regression	51.62	41.65	0	0	20.42
Test SVM	0	100	0	0	0
Train SVM	0	100	0	0	0

The poor performance on curveballs makes some sense, as Alcantara very rarely throws these pitches, peaking at 9.2% in 2018 and down to 0.3% frequency in 2022. The poor performance on sliders, however, is more curious, as he has tended to throw these pitches around 22% of the time for his career. It seems reasonable to attribute the poor performance here to a lack of more robust data and a more sophisticated model would be able to identify the distinction between sliders and other pitches. We can see that ther slight increase in prediction variety is borne out in the prediction vectors. Looking at the same testing data points as above, both models react slightly more to adjustments in data:

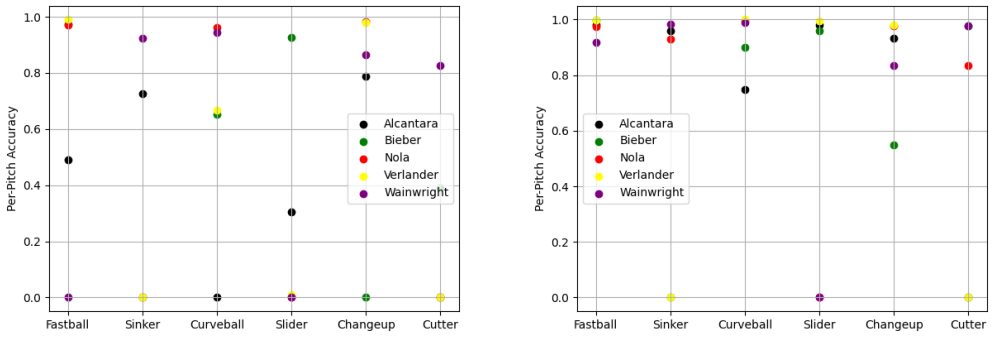
True Label	FF%	SI%	CU%	SL%	CH%
FF	0.306	0.282	0.084	0.204	0.123
SL	0.264	0.290	0.011	0.221	0.213
SL	0.285	0.278	0.028	0.221	0.188
SI	0.286	0.275	0.066	0.219	0.154
FF	0.319	0.291	0.037	0.209	0.151

True Label	FF%	SI%	CU%	SL%	CH%
FF	0.289	0.262	0.090	0.217	0.141
SL	0.265	0.275	0.044	0.219	0.196
SL	0.269	0.241	0.048	0.236	0.205
SI	0.271	0.263	0.060	0.222	0.184
FF	0.293	0.256	0.062	0.219	0.177

The good news? None of these problems occur when we switch our goal to pitch *recognition*:



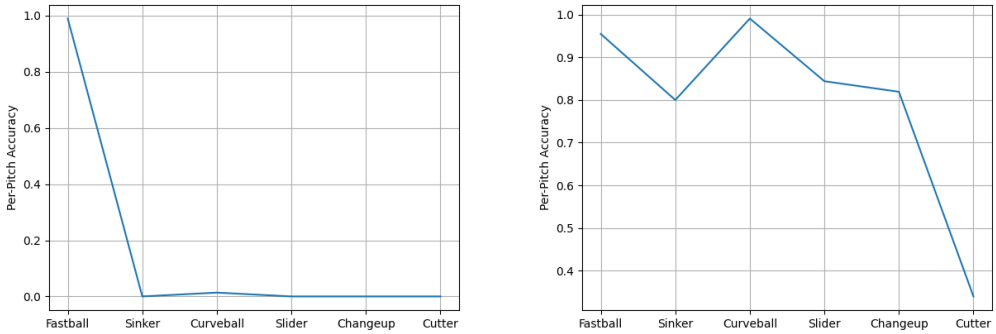
Why is this increased accuracy the case? Simply put, better data. We have access not only to the velocity of each pitch (which by itself is often enough to recognize the correct pitch type) but also spin data, location, and many other metrics that are highly useful for identifying pitches correctly. To illustrate this, let's look at the per-pitch accuracy for SVM and Regression, respectively:



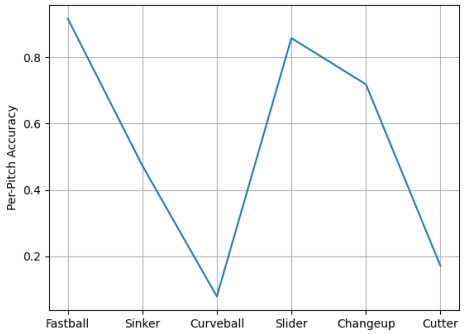
Note that the 0's in the regression graph correspond to pitches that the respective player does not throw (Verlander does not throw a sinker or cutter, so we represent these as 0's on the graph). Again, regression is the star. It only has small hiccups on pitches that are rarely thrown, most notably Bieber's changeup (2.3% usage in 2022) and Alcantara's curveball (0.3% usage in 2022). One would hope that a larger dataset could significantly improve the accuracy of identifying these pitches.

At this point, it is clear that Linear Regression is well-suited to this task, while Support Vector Machines are not. Can we give regression a more general task? I concatenated the datasets for each pitcher and removed all immediate identifying features (pitcher ID's, etc) and ran the regression model on this new dataset. How did it perform?

On prediction, regression had test and training accuracies of 33.32% and 32.86%, which came entirely from its recognition of fastballs. Recognition performs as expected, albeit slightly worse than on a per-pitcher basis, with a training accuracy of 87.33% and test accuracy of 87.27%. This tracks with the previously established literature, which has shown convincingly that effective pitch recognition and prediction relies heavily on knowing the metrics of the specific player that is throwing the pitch we are recognizing/predicting. Note its lack of ability to identify cutters:



Cutters often have similar characteristics to fastballs and so it is not surprising that a simple model would have trouble differentiating the two, especially without knowing the specific pitcher. The model also gets worse when we expand our scope to more pitchers. I gathered pitch data for every single pitch thrown in MLB between June 1 and June 10, 2022, removing every pitch that was not one of the six we have been testing for previously. Running regression on this new dataset, we achieve a very nice 69% accuracy on testing and a less-nice 68.49% on training, with the following per-pitch accuracies:



The model retains its performance on fastballs, sliders, and changeups, but performs significantly worse on sinkers and curveballs.

5 Conclusion

Ultimately, this project was a lesson in understanding the usefulness of certain types of data. We saw that in the pitch prediction model, we simply need a larger variety of data, and possibly more complicated models, to achieve anything beyond blind guessing. In order to perform large-scale accurate *predictive* machine learning on baseball datasets, it seems that simply more data and fire-power is required than what I felt comfortable working with. Especially in the case of time-series data, which I ended up not even being able to play around with too much, the scope of this kind of project requires much more advanced tools than what I have access to. That being said, we were able to have quite accurate pitch recognition models, and contribute a new analysis to the field in the form of multiclass prediction over binary prediction.

References

[1] B. Lindbergh T. Sawchik. *The MVP Machine*. Basic Books, 2019 (cit. on p. 1).

[2] Jake Mailhot. *How Much Did the Astros Really Benefit From Sign-Stealing?* 2019 (cit. on p. 1).

[3] Andy Martino. *Cheated. The Inside Story of the Astros Scandal and a Colorful History of Sign Stealing*. Random House, 2021 (cit. on p. 1).

[4] H. Tran P. Hoang M. Hamilton. "Applying machine learning techniques to baseball pitch prediction". *International Conference on Pattern Recognition Applications and Methods* (2014) (cit. on pp. 1–3).

[5] H. Tran G. Sidle. "Using multi-class classification methods to predict baseball pitch types". *Journal of Sports Analytics* (4 2018) (cit. on p. 1).

[6] J Morehouse. *No Pitch is an Island: Pitch Prediction With Sequence-to-Sequence Deep Learning*. 2022 (cit. on pp. 1, 2).

[7] *Baseball Pitch Prediction*. `towardsai.net/p/machine-learning/baseball-pitch-prediction` (cit. on pp. 1, 3).

[8] James LeDoux. *Introducing pybaseball: an Open Source Package for Baseball Data Analysis*. 2017 (cit. on p. 2).

[9] P. Lawrence. *680 Project*. https://github.com/Paul-Lawrence/project. 2022 (cit. on p. 3).

[10] Ben Clemens. "How Good Are Those Probabilities on the Apple TV+ Broadcasts?" (2022).

[11] Connor Douglas et al. "Computing an Optimal Pitching Strategy in a Baseball At-Bat" (2021).

[12] Neil Vigdor. "The Houston Astros' 2017 Cheating Scandal: What Happened". *New York Times* (2022).