Predicting Pitches

David Buehler

Introduction

- Baseball is like a chess match between pitcher and batter
- Baseball has started to adopt analytics tools like PITCHf/x, Statcast, and Sabermetrics for use to gather and analyze data
- Kaggle had a dataset with four years worth of pitch data from
  - Every pitch thrown in the MLB from 2015-2018
- Wanted to know what factors best determined what pitch was coming next

Who's Interested?

- Managers, hitting, and pitching coaches
  - Also the players themselves
- Any leg up they can get on the competition is the goal here
- If hitters could have some certainty on what pitch was coming next based on count, runners on, pitcher, etc. they would have an advantage
- Model could be beneficial for baserunners too
  - If they think an off-speed pitch is coming, they could get an early jump to try to steal a base

Data Acquisition and Wrangling

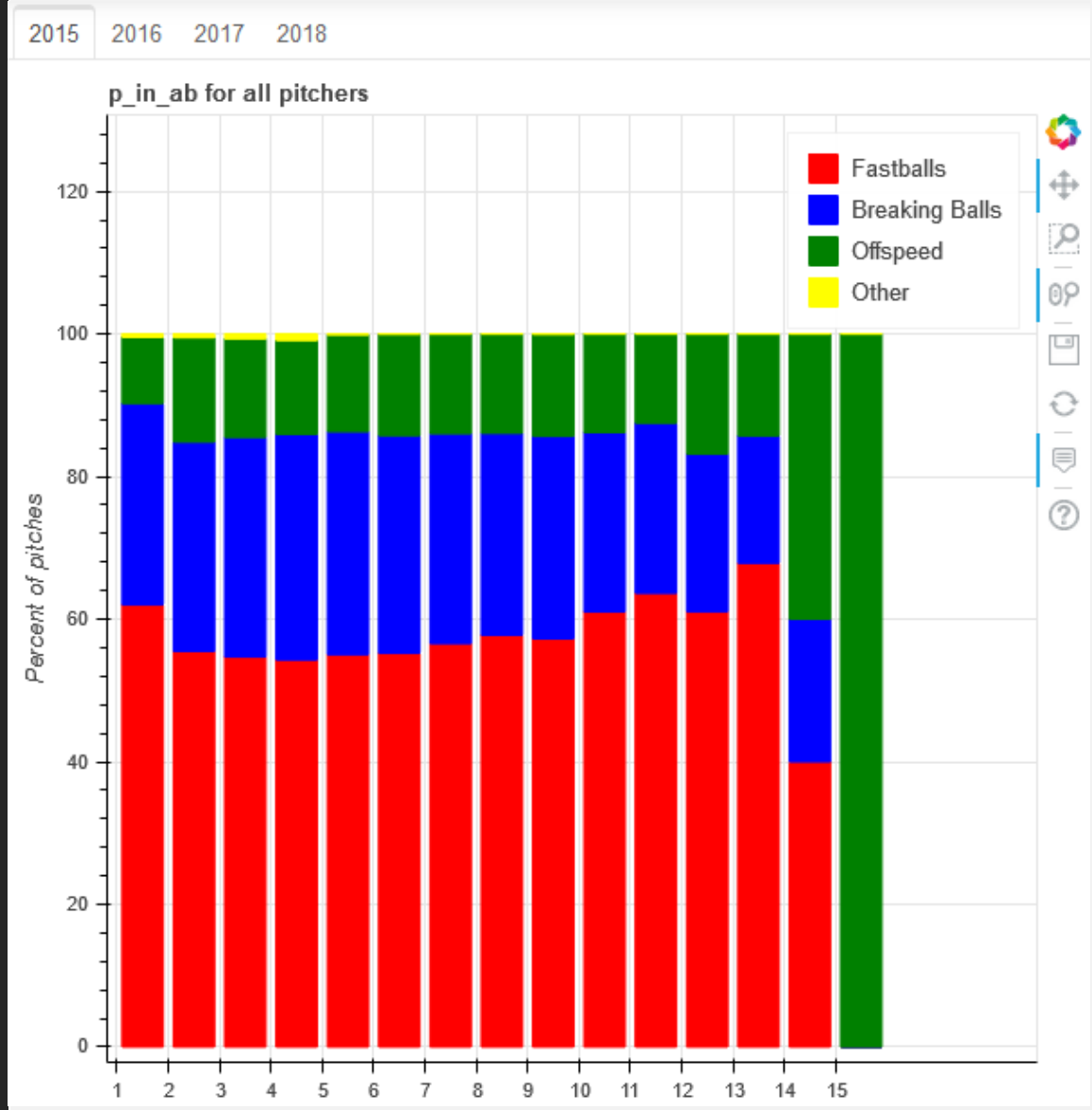
- Data pulled from a Kaggle repository
- Data came mostly clean, though a few straggling data points that needed removing
  - Found a few pitches with a ball count of four, which is not allowed in baseball
- Wanted to find out what pitch in the at-bat each pitch was
- Grouped pitches into four groups: fastballs, breaking balls, off-speed pitches, and other pitches
- Had to merge two data sets to find which pitcher threw each pitch
  - One data set had ID numbers and their corresponding player
  - Simply merged the two data sets
- Combined all four yearly data sets into one large combined data set

Data Storytelling

- Used the Bokeh library for its powerful visualization tools
- Broke up analysis by pitcher and year
- Created functions to pull out how many, and the percentage of fastballs/breaking balls/off-speed/other pitches a certain pitcher threw based on a variable of interest
  - Variables included ball count, strike count, runners on, pitch in at bat, inning, and whether or not the pitcher was ahead in the count
- Looked at Cy Young award winning pitchers from 2015 and 2018

Example Bokeh Graph

—



Inferential Statistics

- Null Hypothesis: "On average, pitchers will throw the same amount of fastballs/breaking balls/offspeed pitches no matter the score differential“
- Created a score differential column, then used a bootstrap test with a 95% confidence interval to determine significance
- Broke up the test by pitch to look at types of pitch individually

Pitch	Mean	95% Min Interval	95% Max Interval	P-value
Fastball	0.0188	0.0139	0.0239	1.3872e-110
Breaking Ball	0.1033	0.0970	0.1099	1.0425e-281
Off-speed	0.1270	0.1171	0.1373	6.3441e-122
Other	1.1583	1.1109	1.2049	0

Inferential Statistics (cont.)

- All came back with p-values less than 0.05
- Means run differential is statistically significant in predicting the next pitch
- Other category
  - Pitchouts which lead to intentional walks are in this category
  - Pitchers don’t want to pitch to good hitters in close game situations
  - Average score was around 1, which means a lot of these pitches happened in very close games

Baseline Modeling

Extended Modeling

Neural Networks

Modeling

Baseline Modeling

- Started off with logistic regression and ridge classifier models as a baseline
- Used GridSearchCV to find the best C value and alpha value
  - C for logistic regression
  - Alpha for ridge classifier
- Modeled entire data set to see how the MLB did as a whole
- Looked deeper into specific pitchers
  - Felix Hernandez
  - Justin Verlander
  - Clayton Kershaw
  - Edwin Diaz
  - Aroldis Chapman

Baseline Modeling Results

- Model accuracies:
  - Logistic Regression: 55.61%
  - Ridge Classifier: 55.56%
- Neither model predicted any off-speed or other pitches
  - Only predicted fastballs and breaking balls, the two largest classes
- Classification Reports:

Ridge Classifier Classification Report:

	precision	recall	f1-score	support
BB	0.42	0.09	0.14	225509
FB	0.57	0.94	0.71	399169
OS	0.00	0.00	0.00	87200
OT	0.00	0.00	0.00	1925
accuracy			0.56	713803
macro avg	0.25	0.26	0.21	713803
weighted avg	0.45	0.56	0.44	713803

Extended Modeling

- Tested three ensemble models and one neighbors model to determine which to delve deeper into
  - Gradient Boosting
  - Ada Boosting
  - Random Forest
  - K Nearest Neighbors
- Gradient boosting performed the best on a sample of my data set
- Performed hyper-parameter tuning on the model to get the best performance
- Looked at feature importance, as well as classification report for model performance
- Used SMOTE over-sampling to reign in class representation between the three classes

Extended Modeling Results

- 10,000 Sample:

Random Forest Classification Report:

	precision	recall	f1-score	support
BB	0.39	0.29	0.33	792
FB	0.59	0.74	0.66	1402
OS	0.14	0.06	0.09	306
accuracy			0.52	2500
macro avg	0.37	0.37	0.36	2500
weighted avg	0.47	0.52	0.48	2500

Gradient Boosting Classification Report:

	precision	recall	f1-score	support
BB	0.42	0.19	0.26	792
FB	0.58	0.87	0.69	1402
OS	0.27	0.01	0.02	306
accuracy			0.55	2500
macro avg	0.42	0.36	0.32	2500
weighted avg	0.49	0.55	0.47	2500

Ada Boosting Classification Report:

	precision	recall	f1-score	support
BB	0.45	0.13	0.20	792
FB	0.57	0.93	0.71	1402
OS	0.00	0.00	0.00	306
accuracy			0.56	2500
macro avg	0.34	0.35	0.30	2500
weighted avg	0.46	0.56	0.46	2500

K Neighbors Classification Report:

	precision	recall	f1-score	support
BB	0.37	0.25	0.30	792
FB	0.58	0.81	0.68	1402
OS	0.00	0.00	0.00	306
accuracy			0.53	2500
macro avg	0.32	0.35	0.32	2500
weighted avg	0.44	0.53	0.47	2500

- 100,000 Sample:

Random Forest Classification Report:

	precision	recall	f1-score	support
BB	0.37	0.30	0.33	7920
FB	0.58	0.71	0.64	14018
OS	0.19	0.09	0.12	3062
accuracy			0.50	25000
macro avg	0.38	0.37	0.36	25000
weighted avg	0.47	0.50	0.48	25000

Gradient Boosting Classification Report:

	precision	recall	f1-score	support
BB	0.44	0.10	0.16	7920
FB	0.57	0.94	0.71	14018
OS	0.26	0.00	0.00	3062
accuracy			0.56	25000
macro avg	0.42	0.35	0.29	25000
weighted avg	0.49	0.56	0.45	25000

Ada Boosting Classification Report:

	precision	recall	f1-score	support
BB	0.41	0.07	0.12	7920
FB	0.57	0.95	0.71	14018
OS	0.00	0.00	0.00	3062
accuracy			0.56	25000
macro avg	0.32	0.34	0.28	25000
weighted avg	0.45	0.56	0.44	25000

K Neighbors Classification Report:

	precision	recall	f1-score	support
BB	0.37	0.25	0.30	7920
FB	0.58	0.81	0.67	14018
OS	0.25	0.01	0.02	3062
accuracy			0.53	25000
macro avg	0.40	0.36	0.33	25000
weighted avg	0.47	0.53	0.47	25000

Gradient Boosting Results

- Used GridSearchCV to find the best n_estimators parameter
- Best model accuracy thus far
  - 56.26%
- Classification Report

Gradient Boosting Classification Report:

	precision	recall	f1-score	support
BB	0.47	0.09	0.15	225509
FB	0.57	0.95	0.71	399170
OS	0.63	0.00	0.00	87199
accuracy			0.56	711878
macro avg	0.56	0.35	0.29	711878
weighted avg	0.54	0.56	0.45	711878

- Good job predicting the majority class, not as well predicting the minority classes
- Top 5 most important features
  - Pitcher side
  - Strike count
  - Ball count
  - Inning
  - Pitcher behind
- Key findings
  - Pitcher side the most important feature
  - EDA backs up what features are more important here
  - Ball/strike count being important to predicting pitches

Over-sampling Results

- Used a 100,000-sample data set
  - Save on time, gradient boosting model wasn't much different between a sampled data set and the whole thing
- Looked at three different levels of class representation
  - 75% breaking ball, 50% off-speed
  - 87.5% breaking ball, 75% off-speed
  - Even split of three classes
  - Percentages based on majority class (fastballs)
- Each model got continually better at predicting the minority classes
- Progressively worse at predicting majority class
- Pitcher side remained most important feature, ball count and strike count dropped off in importance
- Top 5 most important features for all oversampling models
  - Pitcher side
  - Batter side
  - Top of inning
  - On2b
  - Inning

Comparison of Classification Reports

- 75% breaking ball/50% off-speed

Gradient Boosting Classification Report

	precision	recall	f1-score	support
BB	0.43	0.25	0.32	7920
FB	0.59	0.82	0.68	14018
OS	0.26	0.05	0.08	3062
accuracy			0.55	25000
macro avg	0.42	0.38	0.36	25000
weighted avg	0.50	0.55	0.49	25000

- Even split

Gradient Boosting Classification Report

	precision	recall	f1-score	support
BB	0.41	0.38	0.39	7920
FB	0.62	0.59	0.60	14018
OS	0.21	0.31	0.25	3062
accuracy			0.49	25000
macro avg	0.41	0.42	0.42	25000
weighted avg	0.50	0.49	0.49	25000

Neural Networks

- Modeled the data with two different neural networks
  - One from Scikit-Learn, the other from Keras and TensorFlow
- Used a 1,000,000-sample data set
  - Save on time
- Scikit-Learn MLP Classifier was a more basic neural network
  - Used 8 hidden layers with 15 nodes each (15 nodes for 15 variables)
- Keras/TensorFlow model was more advanced
  - Used three Dense layers
  - One for input with a relu activation
  - One for hidden layers, 15 of them also with relu activation
  - One for output, 3 outputs with a softmax activation
  - Used categorical crossentropy for the loss function and the adam optimizer
- Used resampling techniques on both neural networks to try to improve the models

MLP Classifier Results

- Nearly the same as the gradient boosting model in terms of model performance
- Slightly more accurate at 56.20%
- Classification Report

MLP Classifier Classification report

	precision	recall	f1-score	support
BB	0.47	0.10	0.16	79195
FB	0.57	0.95	0.71	140182
OS	0.00	0.00	0.00	30623
accuracy			0.56	250000
macro avg	0.35	0.35	0.29	250000
weighted avg	0.47	0.56	0.45	250000

- Sampling techniques yielded similar results as the gradient boosting model
  - Each model got continually better at predicting the minority classes
  - Progressively worse at predicting majority class

MLP Classifier Over-sampling Classification Reports

- **66.667% breaking ball, 50% off-speed**

MLP Classifier Classification report

	precision	recall	f1-score	support
BB	0.44	0.23	0.31	79195
FB	0.59	0.78	0.67	140182
OS	0.24	0.19	0.21	30623
accuracy			0.53	250000
macro avg	0.42	0.40	0.40	250000
weighted avg	0.50	0.53	0.50	250000

- **Even split**

MLP Classifier Classification report

	precision	recall	f1-score	support
BB	0.41	0.46	0.43	79195
FB	0.67	0.41	0.51	140182
OS	0.20	0.48	0.28	30623
accuracy			0.44	250000
macro avg	0.42	0.45	0.41	250000
weighted avg	0.53	0.44	0.46	250000

Keras Model Results

- Slightly less accurate at 56.07%
- Slightly different predictions than either the gradient boosting model and MLP classifier
  - More predicting fastballs, less on breaking balls
- Classification Report

	precision	recall	f1-score	support
0	0.56	0.99	0.72	140182
1	0.48	0.02	0.04	79195
2	0.00	0.00	0.00	30623
accuracy			0.56	250000
macro avg	0.35	0.34	0.25	250000
weighted avg	0.47	0.56	0.41	250000

- Sampling techniques yielded similar results as the gradient boosting and MLP classifier model
  - Each model got continually better at predicting the minority classes
  - Progressively worse at predicting majority class
  - Fluctuates more than previous models

Keras Over-sampling Classification Reports

- 75% BB, 50% OS

	precision	recall	f1-score	support
0	0.60	0.68	0.64	140182
1	0.41	0.43	0.42	79195
2	0.25	0.05	0.09	30623
accuracy			0.53	250000
macro avg	0.42	0.39	0.38	250000
weighted avg	0.49	0.53	0.50	250000

- 87.5% BB, 75% OS

	precision	recall	f1-score	support
0	0.66	0.39	0.49	140182
1	0.39	0.53	0.45	79195
2	0.20	0.40	0.27	30623
accuracy			0.43	250000
macro avg	0.42	0.44	0.40	250000
weighted avg	0.52	0.43	0.45	250000

- Even Split

	precision	recall	f1-score	support
0	0.68	0.35	0.47	140182
1	0.41	0.45	0.43	79195
2	0.19	0.57	0.29	30623
accuracy			0.41	250000
macro avg	0.43	0.46	0.39	250000
weighted avg	0.53	0.41	0.43	250000

Conclusion

- Pitcher side, ball count, strike count, batter side all seem to be the best predicters of pitches
  - Different handed pitchers will have different strategies based on batter handedness
  - Pitch selection will vary based on count as well.
  - A pitch thrown on an 0-2 count will usually be different than a 3-0 count
- Inning was also an important factor in the models
  - As the game goes on, pitch selection could change and adapt
  - Relief pitchers come in and have different pitches and pitch selection than the starting pitcher

Future Work

- Delve deeper into the deep learning neural networks and extract feature importance from them. I was unable to get feature importance from the deep learning models, so getting more information in that regard could tell us more about what goes into predicting pitches
- Adapting these models for real time data and adjusting hyperparameters accordingly could be a good way to use these models for use during a game. Using one of these models to predict what pitch is coming next was the point of building them, so putting them to use during a game would serve their intended purpose.