Predicting Pitch Outcome

How to calculate the outcome of the next pitch using contextual game data and a neural network.

Bryce Lund, Eric Louis, Jeffrey Mohler, Gage Poulson

Anyone who knows sports knows that the game of baseball is rich in data. Analytics and modeling have been used for years to help teams and organizations gain a competitive advantage and, in turn, more profit. For this reason, statistics and models are abundant. As of late, pitch-level analytics and models have been the theme. For example, great research has been put into predicting the next pitch (see here and here). These models see the game from the hitting team's perspective and aim to help the offensive team know what is going to happen next and prepare for it.

In our project, we chose to come at the question from a different angle. We chose to look at the question from the defensive or pitching team's perspective. In short, how can we improve pitch selection? Currently in Major League Baseball, pitches are most often selected by catchers with the help of scouting reports and their individual game knowledge among other things. But, what if there was a machine learning model that, at least, gave some support for what pitch is called?

Hypothetically, this model could be used, in the dugout, from a coaches iPad as a tool in the decision making process. Or, it could be used as a tool to enhance the experience of the broadcaster or viewer. We all know that we like to feel like we know better than the players! This webapp is the prototype our team came up with after completing this project. We used python and a neural network to predict the best pitch to throw when desiring a certain outcome. For the webapp, we used a React frontend with a Node.js backend and hosted it on Heroku. Below, you can dive deeper into the details of our process and thinking.

The Data

We used a python package called pybaseball which scrapes baseball-reference.com and baseballsavant.com enabling us to retrieve statcast data, pitching stats, batting stats, and division standings/team records easily.

Pybaseball is a python package that can be used to easily and consistently bring in baseball data into python. The pybaseball github repository can be found here. Multiple functions can be found in the package, but we chose to use the statcast function, which provides pitch by pitch data across a certain date range. A specific team can also be specified as a parameter. For this project, we pulled one season of baseball data. In a true machine learning environment, this statcast function could easily use relative date variables so that data could be consistently updated. However, because baseball is not currently being played (COVID-19), we chose to train the model on a single season for our prototype.

This statcast function not only provides current contextual game data but also specific data relative to each pitch. This data includes variables such as release speed, break angle, and the coordinates of the ball as it crosses home plate. While these fields are interesting and helpful in many cases, we chose to exclude them for our current purposes. With our goal of predicting which pitch to throw, we could only use data available up until the point of the pitch. This date consisted mainly of contextual data about the game or at bat (score, balls, strikes, pitcher and batter stance). A full list of the variables used in the model can be found to the below.

pitch_type	inning	on_base
description	at_bat_number	p_throws
balls	pitch_number	inning_topbot
strikes	home_score	if_fielding_alignment
outs_when_up	away_score	of_fielding_alignment
stand	home_team	away_team

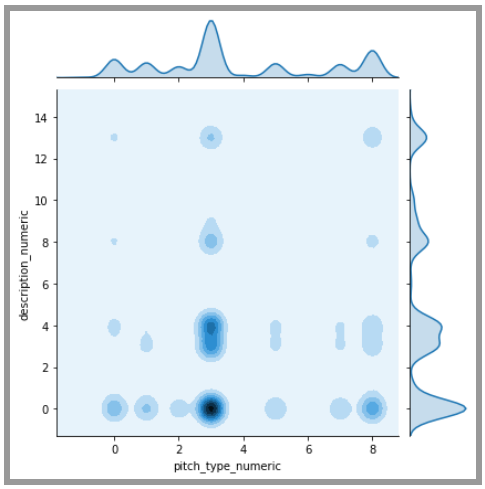
The data also provides us with the outcome of each pitch thrown. The 15 possible outcomes are shown to the right. While the outcome cannot be known before the pitch is thrown, we chose to keep it in our model. We did this so that the predicted pitch could be based upon what the user wanted the outcome to be. In other words, the model takes current game information along with the desired outcome and, in response, returns the pitch most likely to result in that desired outcome.

```
called_strike
ball
hit_into_play_score
swinging_strike
hit_into_play
foul_tip
hit_into_play_no_out
foul
swinging_strike_blocked
blocked_ball
foul_bunt
pitchout
hit_by_pitch
missed_bunt
bunt_foul_tip
```

The Cleaning and Exploration

Cleansing and exploration of the data was reduced due to the clean data that pybaseball provided. Yet, some tasks were still necessary.

With the data in hand, we were able to explore the data and see what adjustments would need to be made. Gratefully, the data came in very nicely from the pybaseball package. Still, to start our cleansing process, we checked and visualized various univariate and bivariate stats. This led us to some of the changes made as well as some interesting facts.



For example, the heatmap to the right shows pitch type on the x-axis and description (outcome) on the y-axis. The really dark area in the bottom middle refers to pitching a fastball that results in a ball. This, of course, makes sense for baseball because the majority of pitches are fastballs. Looking into this also helped us realize that some pitches were thrown less than 50 times in 750,000+ samples. We, then, reduced our data to the 9 most common pitches thrown as the others would not make for accurate predictions.

.

One of the other major cleansing tasks we did was converting all player ids to boolean fields (see below). The data came in from baseball with player ids in several fields giving information into who was on base, who was pitching, who was in the field, etc. It seemed like a great idea to train the model very specifically down the players at first. However, after looking into it, we decided that this was out of the scope of this project. So, we converted all of these fields to booleans, so that we could still know if there was anyone on base for instance. Still, using player ids are something that can and should be looked into in a later project.

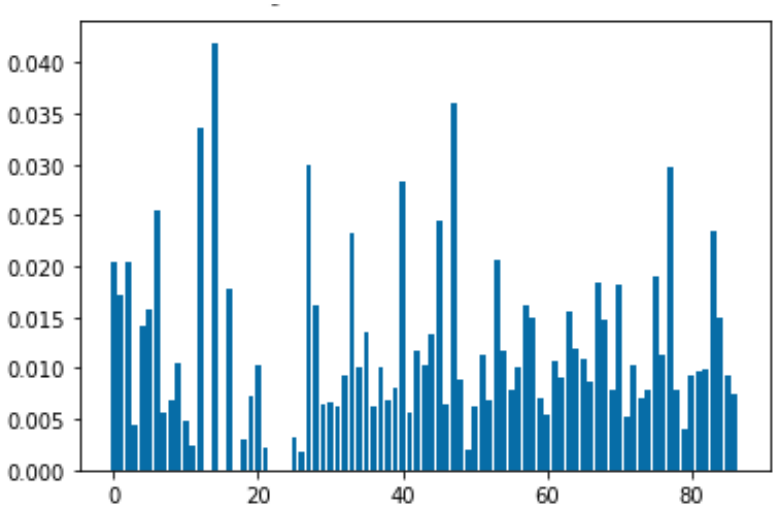
```
df['on_1b_bool'] = df.on_1b.apply(lambda x: False if x == '-' else True)
df['on_2b_bool'] = df.on_2b.apply(lambda x: False if x == '-' else True)
df['on_3b_bool'] = df.on_3b.apply(lambda x: False if x == '-' else True)
```

The final major step we took in our cleaning was transforming all of the categorical variables into dummy variables. In this process, we obviously realized that the majority of our variables were categorical. This not only impacted our choice of a model (as stated below), but it also affected how we performed the remainder of cleansing. For example, we did not need to correct for skewness or kurtosis on these categorical variables.

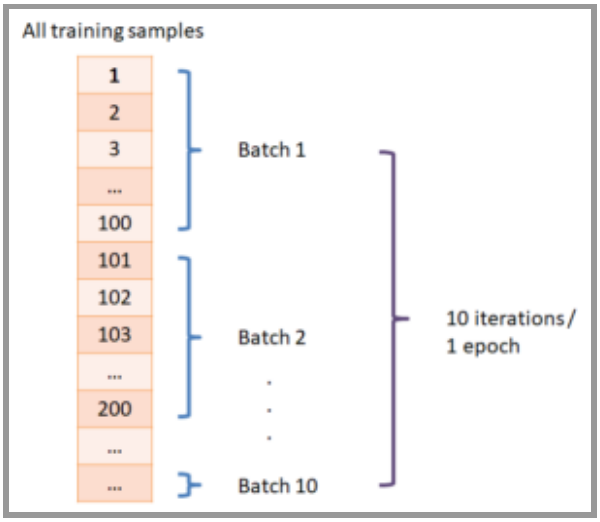
The Model

Keras is a neural network capable of running on tensorflow. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Choosing our model was one of the key steps we took in our process. We not only had several variables, but many of these were categorical. This caused us to have many more when transforming them into dummy variables. As part of our process, we tested several models. For example, we tested an XGBoost model. On our first run, the accuracy was around 5%. In addition, none of the features had that great of an impact on the model. The graph to the right shows the feature importance of this model. With all of the features having relatively low importance, we decided to use a model that would be better suited to finding complex relationships and patterns between the variables.



For this reason, we decided to use a neural network built on Keras. Neural networks allow us to find complex relationships between variables in our input that may not always be immediately apparent. We trained our model on 150 epochs with a batch size of 256. So what does that all mean? Batch size is the number of samples to work through before the model adjusts any of the parameters. All the data is run through the model 150 times to repeatedly expose the model to the data and more closely fit the model to the data over each pass. The relationship of epochs and batch size is like nested for loops. The outer for loop is the epoch, meaning all the data is processed 150 times, the inner for loop is the batch size, in this case running 128 samples before adjusting parameters. These two hyperparameters can be adjusted to produce the best overall accuracy.



Our model consists of an input layer, 6 hidden layers, and an output layer. Our hidden layers scale from 300 nodes down to 100 nodes, each with a varying degree of dropout. Dropout layers randomly select a percentage of nodes to be disabled during each epoch. This method was used to regularize the network and help it better generalize results on new data. Dropout was chosen as it is more computationally efficient and better suited to this problem than other regularization methods such as L1 or L2. Because this is a classification problem, softmax was used as the activation function for our output layer to make sure just one of the options was selected for each prediction.

```
classifier = Sequential()
classifier.add(Dense(units=87, activation='relu', input_dim=87))
classifier.add(Dropout(0.1))
classifier.add(Dense(units=300, activation='relu'))
classifier.add(Dropout(0.3))
classifier.add(Dense(units=250, activation='relu'))
classifier.add(Dropout(0.25))
classifier.add(Dense(units=100, activation='relu'))
classifier.add(Dropout(0.1))
classifier.add(Dense(units=100, activation='relu'))
classifier.add(Dropout(0.1))
classifier.add(Dense(units=13, activation='softmax'))

classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

classifier.fit(X_train, y_train, epochs=250, batch_size=256, validation_data=(X_test,
y_test), shuffle=True)
```

The Results

Overall the model achieved an accuracy of 41%. This is much better than an 11% accuracy of a random guess of the 9 different outcomes.

In order to properly understand the results it is important to learn a few new vocabulary words.

Precision: Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives. Said another way, “for all instances classified positive, what percent was correct?”

Recall: Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. Said another way, “for all instances that were actually positive, what percent was classified correctly?”

F1 Score: The F_1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F_1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F_1 should be used to compare classifier models, not global accuracy.

Support: Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn’t change between models but instead diagnoses the evaluation process.

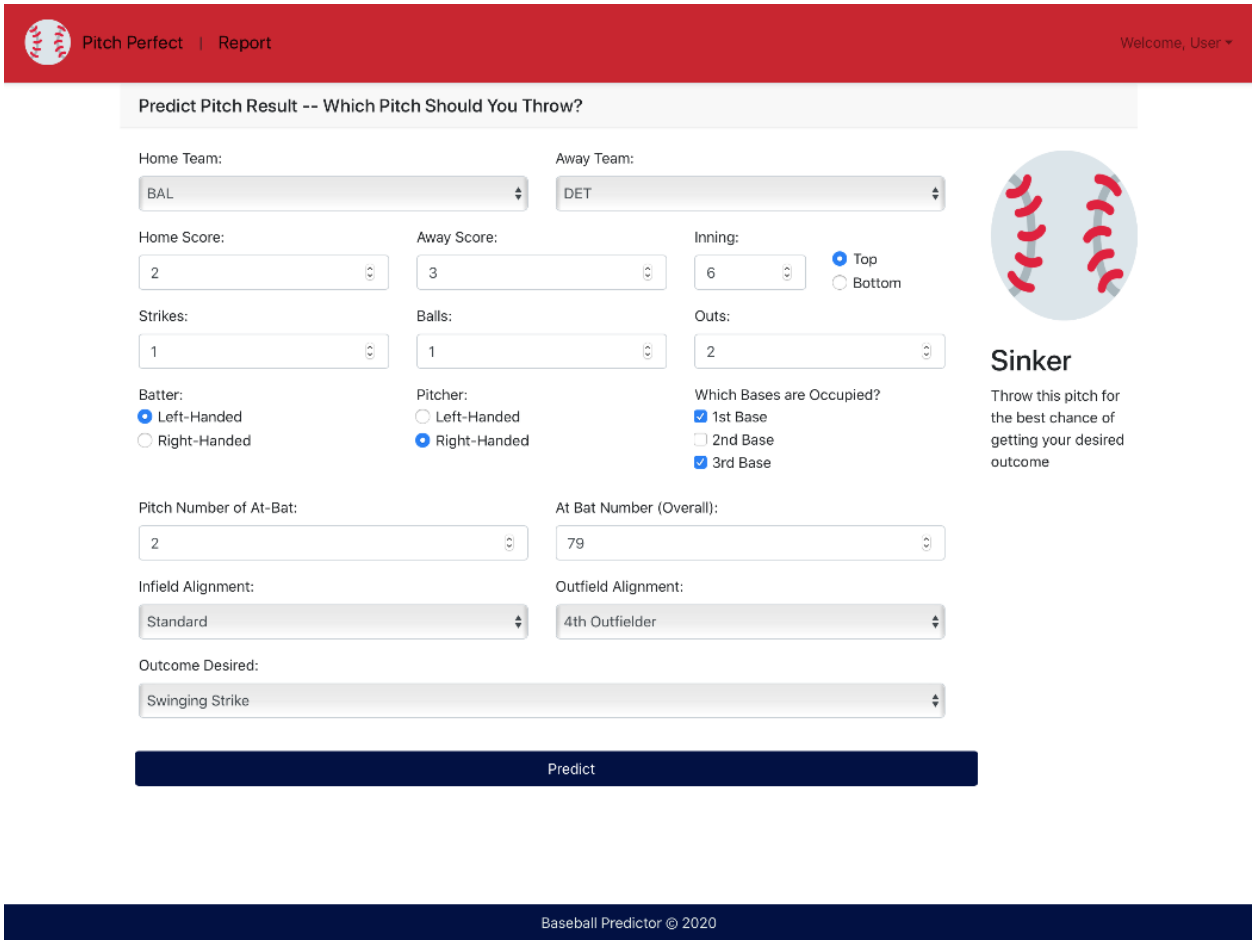
Below is the classification report for our model. It looks at the 9 possible pitch outcomes and rates them in the metrics discussed above. As you can see, this neural network performed much better than the XGBoost model returning a 41% overall accuracy.

**note - the following tables are the same. Only kept both for options with formatting when putting on blog

```
                precision    recall  f1-score   support

         Slider       0.52      0.01      0.02      6655
4-Seam Fastball       0.56      0.01      0.02      6539
       Changeup       0.65      0.09      0.15      6674
      Curveball       0.66      0.01      0.01      6591
         Sinker       0.70      0.40      0.51      3626
2-Seam Fastball       0.57      0.10      0.16      6539
  Knuckle Curve       0.65      0.34      0.45      5177
   Split Finger       0.62      0.15      0.24      6616
   Knuckle Ball       0.51      0.02      0.03      6566

      micro avg       0.64      0.10      0.18     54983
      macro avg       0.60      0.12      0.18     54983
   weighted avg       0.60      0.10      0.15     54983
    samples avg       0.10      0.10      0.10     54983

overall accuracy      0.41
```

	Precision	Recall	F1-Score	Support
Slider	0.52	0.01	0.02	6655
4-Seam Fastball	0.56	0.01	0.02	6539
Changeup	0.65	0.09	0.15	6674
Curveball	0.66	0.01	0.01	6591
Sinker	0.70	0.40	0.51	3626
2-Seam Fastball	0.57	0.10	0.16	6539
Knuckle Curve	0.65	0.34	0.45	5177
Split Finger	0.62	0.15	0.24	6616
Knuckle Ball	0.51	0.02	0.03	6566
Micro Avg	0.64	0.10	0.18	54983
Macro Avg	0.60	0.12	0.18	54983
Weighted Avg	0.60	0.10	0.15	54983
Samples Avg	0.10	0.10	0.10	54983
Overall Accuracy	0.41			

A site was developed that uses this model. As shown, given the current state of a game, and a desired outcome of a Swinging Strike, our model recommends that the pitcher throw a Sinker on the next pitch.

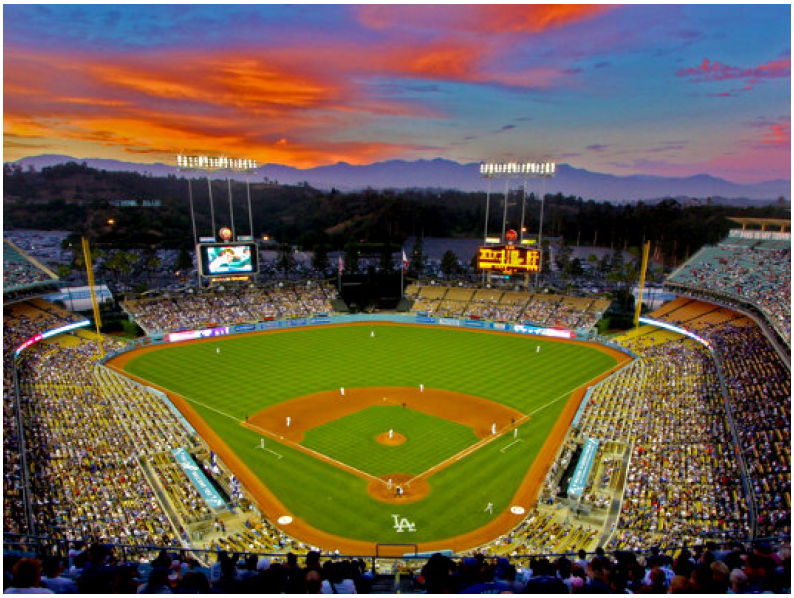


The Conclusion

41% is not 100%, but this is just the start. When will your team start using the data to help in the decision of what pitch to throw? Data doesn't lie.

Going into this project, my team did not know what to expect. Yet, we are pleased with the results that we found. We believe that this is a good start to helping pitchers know what pitch to throw in certain situations. It could also be a good start to helping put machine learning into game time pitch selection. But, the most realistic application is incorporating it into the broadcasting and viewing experience. As mentioned at the start of this article, this model and webapp could be used to help the broadcaster or viewer feel like they know more than the coaches or manager. With this model, viewers will have the opportunity to look at the catcher like Lebron James looks at J.R. Smith in the image below.

We do recognize that our model has its limitations. The model currently does not account for what kind of pitches a certain pitcher can't throw. It also doesn't take into account which players are involved in the at-bat or play. These are things that would likely improve the accuracy and reality of the model. Implementing them with a model made for a specific team may also make it more realistic. Improvement could also be made by grouping together certain pitch outcomes as well. There are many possible outcomes on each pitch and it could be beneficial to group several similar ones together to help train the model and get higher accuracy. Overall, this 8 week project produced better results than we could have ever hoped for!



What our model contributes:

- A pitching (defensive) perspective to next-pitch models
- An enhancement to the viewing and broadcasting experience
- A foundation for future more developed pitch selection models

Future considerations for expanding our model:

- Incorporate player IDs for specific data relating to play
    - Pitcher stats
    - Batter stats
    - Fielding stats
    - Players on base(stealing stats)
- Limit pitches available to model based on the pitcher and their specific pitches
- Group together pitch outcomes or pitch types




Bryce Lund, Eric Louis , Jeffrey Mohler, Gage Poulson
bryce.lund@gmail.com, Static44@gmail.com jeffreydmohler@gmail.com
gagewilliampoulson@gmail.com