Predicting Pitch Location in Professional Baseball Using Neural Networks

Ashna Guliani and Brandon Zink

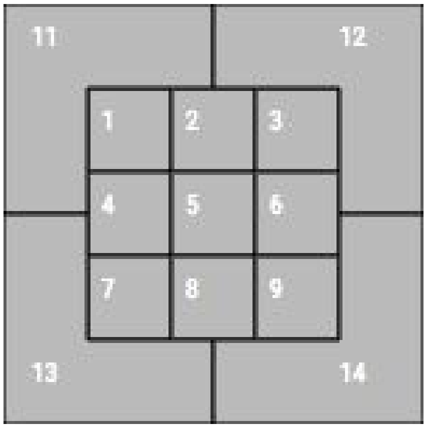
1. Problem

Hitting a baseball is one of the most difficult things to do in sports. If a major league pitcher threw a 90 mph fastball right down the middle of the zone, 95% of the population would look like a young adult confronted with a major life decision. Add in the fact that there a roughly 2 foot by 3 foot area the batter has to cover (called the strike zone, which is the width of home plate and the distance from the batter's knees to the bottom of the letters on their jersey, roughly), and that the pitcher adds all sorts of break and spin to the ball, and it doesn't get any easier.

Our goal is to help simplify this by predicting where the pitch is going to cross the plate. Pitchers generally try to move their pitches around, but everyone is susceptible to some kind of a pattern. Maybe a pitcher likes throwing high in pitches on a 1-2 count against righty batters, or maybe they often throw pitches down and away to lefties [1]. By using a neural network, we can take in a number of factors in attempt to predict where the next pitch is going to be thrown. If this is successful, it becomes extremely advantageous to the hitter if he knows what pitch is coming next. It also would be advantageous to the catcher to understand their tendencies and be able to better study the game.

2. Methods

The strike zone is broken up as follows (1-9 are stikes, 11-14 are balls):



The pitches are tracked via Statcast, which is a (partially) publically available baseball database. Using pybaseball, a python library, this data can be directly imported into a Pandas Dataframe, with extensive information about the situation, attributes, and results of each pitch [2].

We decided to use the following attributes for each pitch:

- The location of the previous three pitches in the plate appearance (PA). A unique identifier was used if it was the first pitch of the PA (location = -1), else the location was 0 if it was the second or third pitch of the PA. This is a number from 1 to 14, excluding 10.
- The count (balls and strikes) in the PA. This is a number from 0 to 3 for balls and 0 to 2 for strikes.
- What handedness the batter is (righty or lefty). This is either ‘R’ or ‘L’.
- The base state, meaning if there are runners on each of the bases. This is three binary categories, one for each base, with true if the base is occupied and NaN otherwise (this is changed to false, or 0).
- The number of outs in the inning. This is a value between 0 and 2.
- The pitch number of the PA. This is similar to the count, but includes foul balls. This is any number from 0 to infinity, though realistically the max is in the 15-ish range, and mode is ~5.

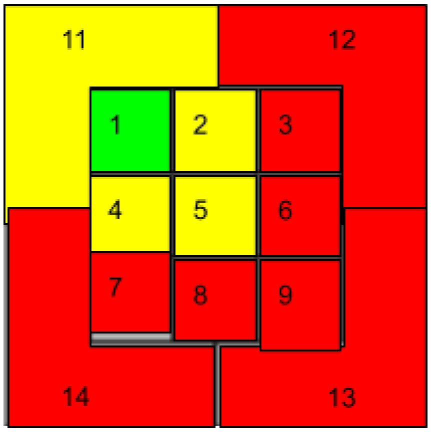
We decided to use a neural network to approach this project. However, we first needed to consider how to organize the data. Keras and sklearn thankfully provide a number of packages to assist in this. Sklearn’s LabelEncoder was used on handedness to turn ‘R’ and ‘L’ into floats. The pitch locations were hot encoded using Keras’ to_categoricals. All of the rest of the data was put through sklearn’s Standard Scalar fit_transform. This means that all of our data is numerical, normalized, and hot encoded when it is categorical in nature [3][4].

In the actual implementation of the neural network, we use Keras’ sequential model with one hidden layer with 64 nodes and ReLU activation. We used the ‘adam’ optimizer and categorical cross entropy loss. We fit the model using a batch size of 5 and an epoch of 50. These decisions were made in reference to a tutorial found online that was used as reference for the implementation of this project [5].

As you use the project, the data is downloaded using pybaseball from Baseball Savant, who allows you to get Statcast data for a specific pitcher. If you have already retrieved that pitchers data before, it will store it in the “Data” folder in .csv format. If you have already run the model for a given pitcher, the model will be saved in the “Data” folder as well in .h5 and .json format to pull the full model. Both downloading the raw data and running the model takes a fair amount of time, so this is to expedite things. If for whatever reason you wish to reset the model or redownload the data, just delete these files and it will automatically download/run the next time you execute the code.

In order to test our results, we used a separate data set from 2018-present (as opposed to 2015-17 for training), performing the same data preprocessing as described above. Using the predict function, it returns a set of probabilities for each zone. By taking the highest probability, we have prediction for where that pitch will be thrown. We can compare that to the actual location.

However, we also care about being close. Picking the exact zone is great, but picking a neighboring zone is also really helpful. For example, refer to the zones picture at the beginning of this paper. Say a pitch is actually thrown in the "1" zone, but we predicted that it was thrown in the "2" zone, then we'll consider that half correct, as it is still helpful to the batter. Using this, if we predict the zone exactly, we consider that a full success for our accuracy metric, if we get a neighboring (or touching) zone, we consider if half a success for our accuracy metric.



If you consider the example above (and my beautiful paint skills), if the pitch were in the "1" zone, we would get a correct for the 1 zone, partially correct for 2, 4, 5, and 11, and incorrect for all the rest. You may also note that there is no 10. That is due to the way Statcast categorizes pitches, not due to our choosing.

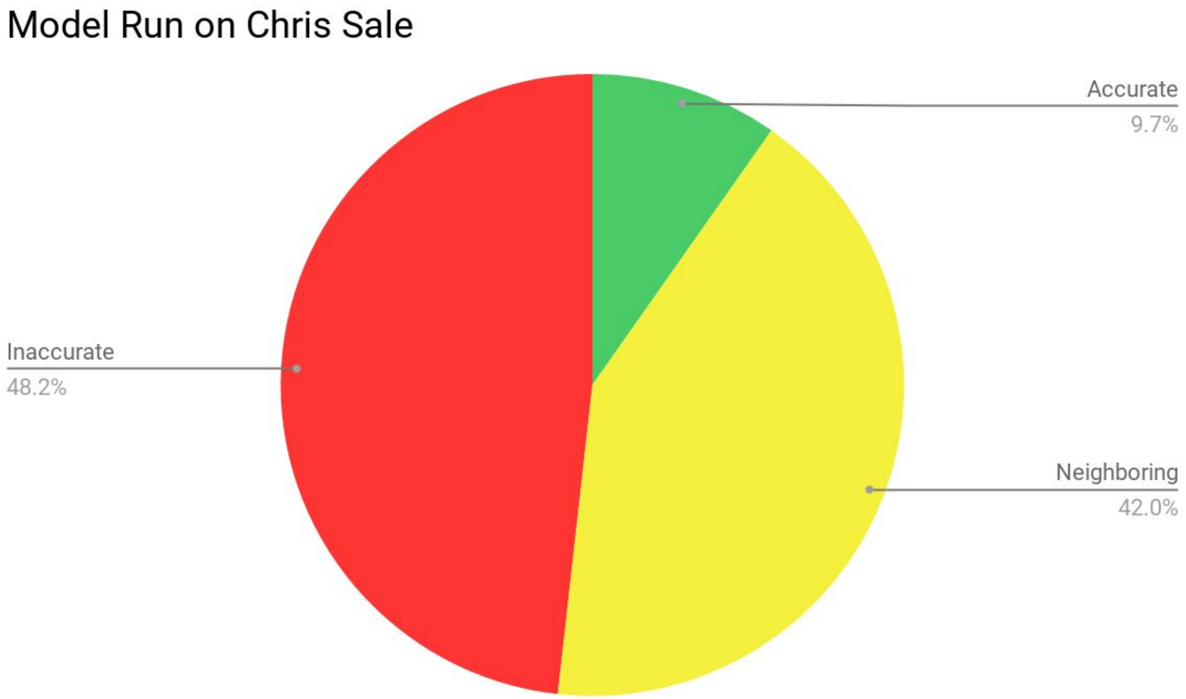
3. Results

Our model, when run for the pitcher Chris Sale, gives an accuracy of 29.5%. This number is found through the method highlighted in the previous section. For example, the first 10 entries of the test set for Chris Sale are shown below.

Actual Pitch Location	Predicted Pitch Location
1	6
13	6
4	6
8	1
4	5
14	6
11	6
12	6
5	5
11	6

The red entries are ones that the model did not predict well. The yellow ones are when the predicted pitch was in a neighboring zone to the actual pitch. The green entries are when the pitch was predicted accurately.

For example, on the first ten pitches of our test set (2018-present), there were 3 neighboring pitches and 1 accurate of 10 total. That would net us an accuracy score of [(1 accurate*1)+(3 neighboring*0.5)]/10 = 25%.



Overall, the model predicts the pitch completely accurately about 10% of the time. An additional 42% of the time, it predicts the pitch to be in a neighboring zone. About 48% of the time, the model is inaccurate.

If we consider the fact that the average MLB plate appearance is 3.9 pitches [6], we can expect to predict the exact location of a pitch once every 2-3 plate appearances, and the general range 1-2 times per plate appearance.

There are, of course, improvements that could be made upon our model given sufficient time and resources. Pitchers consider the batter that they are facing when pitching to them. If the batter is an excellent low ball hitter, the pitcher may consider throwing more high pitches.

It should also be considered that just because a pitch goes somewhere, it doesn’t mean that it was supposed to go there. A catcher may call for a low in changeup, and a pitcher may miss over the heart of the plate. This model would show that they pitch went over the heart of the plate, while that may not be an accurate representation of what the pitcher was actually trying to do.

4. References

[1] Woodward, N. (n.d.). A Decision Tree Approach to Pitch Prediction. Retrieved April 30, 2019, from https://tht.fangraphs.com/a-decision-tree-approach-to-pitch-prediction/

[2] Jldbc. (2019, March 02). Jldbc/pybaseball. Retrieved April 30, 2019, from https://github.com/jldbc/pybaseball

[3] Documentation of scikit-learn. (n.d.). Retrieved April 30, 2019, from https://scikit-learn.org/stable/documentation.html

[4] Keras: The Python Deep Learning library. (n.d.). Retrieved April 30, 2019, from https://keras.io/

[5] Brownlee, J. (2017, June 15). Multi-Class Classification Tutorial with the Keras Deep Learning Library. Retrieved April 30, 2019, from https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/

[6] 2018 Major League Baseball Pitches Batting. (n.d.). Retrieved April 30, 2019, from https://www.baseball-reference.com/leagues/MLB/2018-pitches-batting.shtml