CS230

Predict Next Baseball Pitch Type with RNN

Yifan PI
SCPD
Stanford University
ypi@stanford.edu

Abstract

Baseball players guess the next pitch type will be thrown by pitcher during the game based on the progress of the game, the player statistics, and some human ad-hoc factors. We model the baseball game next pitch prediction as a time series binary classification supervised learning problem. We employ a RNN structure to build a model to accomplish this by learning the memory about previous pitches in the game automatically. We also use a neural network embedding to make player information as an input of the prediction network. We obtain a slightly better performance over the naive prediction.

1 Introduction

Baseball game is a sequence of pitches thrown from pitchers to hitters. One hitter can see usually 1 to 10 pitches from a pitcher before out (earn 3 strikes: did not hit pitches are easy to hit or swing-and-miss, or hit the ball to a location that fielders are able to make a play[1]) or scoring a hit (hit the ball in a location that fielders cannot make a play), or draw a walk. A pitcher tries to throw pitches that a hitter is hard to hit. A strategy for pitcher is to vary pitch types thrown to a particular hitter, so that the hitter have to read the pitch type in the real time, and there is less than 0.2s thinking time for a hitter to decide in the real time[3] . On the other hand, knowing the next pitch type that pitcher will throw can be a huge advantage for a hitter since hitting strategies vary greatly for different pitch types. For example, professional pitcher usually throw fastball 15-20 mph faster than a curve ball, thus resulting a different hit timing.

There are about 13 commonly (e.g., fastball, curve ball, slider etc.) pitch types that pitchers can throw to a hitter. Fastball is the most common pitch type. Most major league pitchers throw it with over 50% of the time. On the other hand, Fastball has a mostly straight travel path comparing to other all other pitch types, thus considering easier to hit if a hitter prepares for it. So predicting whether the next pitch is a fastball or not is a practical problem for baseball hitters.

We define a baseball game as a time sequence of pitches. Then for the particular next pitch prediction problem, this becomes a many-to-many time series prediction problem, i.e., our task is to predict:

$$\Pr[\text{Pitch } t \text{ is a fastball}|\{D_s : s = 0, ..., t-1\}, D'_t]$$

for every $t > 0$ where D_t denotes the data on pitch t, D'_t denotes the data on pitch t but before pitch t is thrown (e.g., the player info and ball/strike count). See also Figure 1 for an illustration.

[1] A play means the hitter hits a pitch to but the hitter or at least one runners on bases are out.

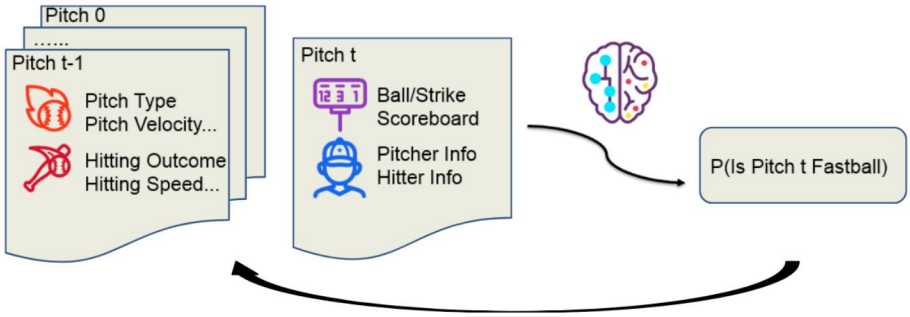

Figure 1: Next Pitch Type Prediction Problem

Unlike other sports, a baseball game can be described well by a structured game log about each pitch. And professional baseball sport leagues like MLB (Major League Baseball) collects data for each game for people analyze. We can find a lot of intuitions that make machine can predict pitch type with data. For example, pitcher is tend to throw fastball when he is behind the count; Not all pitcher can throw all pitch types, a pitcher can throw less types of pitches other than fastball tends to throw more fastball; A pitcher tends to go back to fastball when he throws a lot of balls on other pitches before; A pitcher may challenge a hitter if he sees the hitter is not aggressive on hitting velocity.

2 Related Works

Using quantitative methods in baseball has made a lot of real world successes. The most famous example is [7], which describes how Michael Lewis and his teammates applied data analysis to assembling a competitive baseball team establishing a game winning strike record in MLB history. As pointed out by a survey [6] in 2017, Machine learning has already been used widely in baseball data analysis. However, neural networks techniques were only used in 3/32 research papers the survey covers, which is quite low comparing to the general hotness of neural networks in other ML applications.

[4] encodes player as vectors via neural networks to get interesting results like comparing the style of different players, and making more accurate at bat result prediction than traditional statistic methods. The model was build upon the dummy task from pitcher id and hitter id to the at-bat outcome. We use a similar approach to encode player ids from one-hot vectors to compressed real number vectors. However, the dummy task we use is to predict the probability that a fastball is thrown. [5] uses SVM to predict pitch type. The work fixes last 3 pitches as prior knowledge, and batter profile is consisted by standard statics like slugging percentage.

3 Dataset

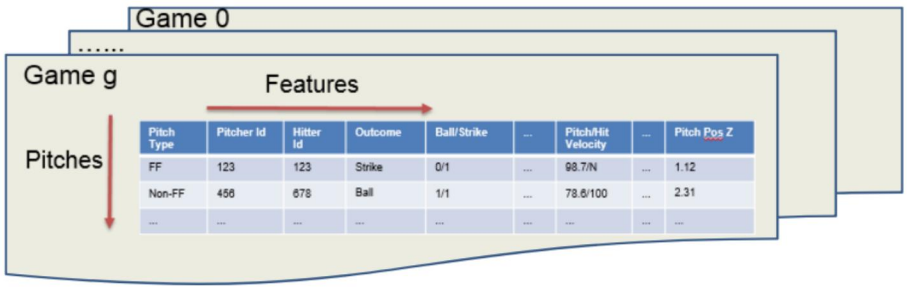

Figure 2: Game by game pitch sequence illustration

MLB started to collect detailed pitch-by-pitch data (called StatCast) from 2008. The dataset consists of ~2500 games per year, and on average ~250 pitches per game (Some extra-inning games can have over 400 pitches). To establish a time series of pitches, we group pitches into games and sort them by the order that the pitch was thrown. For each pitch, we use the following features:

- Player information: See the next section on how we encode player id into an embedding.
- Ball/strike count: Since there can be only 0-4 balls or 0-3 strikes, we encode each of them as a hot-vector as categorical data.
- Pitch/Hit information: Pitch velocity and location, and hit velocity (if any). We normalize these real numbers to $[-1, 1]$ interval by shifting with mean and variance over all data.

We scrap the data via a Python tool [1] to scrap data from [2] for MLB 2014 to 2017 seasons. We train our model on 2014, 2015 and 2016 seasons data (~7500 games in total), and split 2017 season data into 1500 and 1000 games as the dev and the test set. We also truncate each game to first 128 pitches to form uniform length across all games. See Figure 2 as an illustration of data.

4 Methods

The problem is a standard many-to-many time series binary classification problem, thus we use a Recurrent Neural Network to capture previous pitches information by learning. Specifically, we use a Long Short-Term Memory (LSTM) unit as the main model of the predictor. Specifically, for each pitch t, the LSTM unit takes the statistics (e.g., velocity) about the last pitch (zero for the first pitch), and the information before the pitch t is thrown (e.g., pitcher/batter information). We takes the output of the LSTM unit and pass it to a dense neural network (DNN) (with ReLU as activations between hidden layers) and takes Sigmoid output as the prediction of the probability that pitch t is a fastball. See Figure 3 for an illustration of the network.

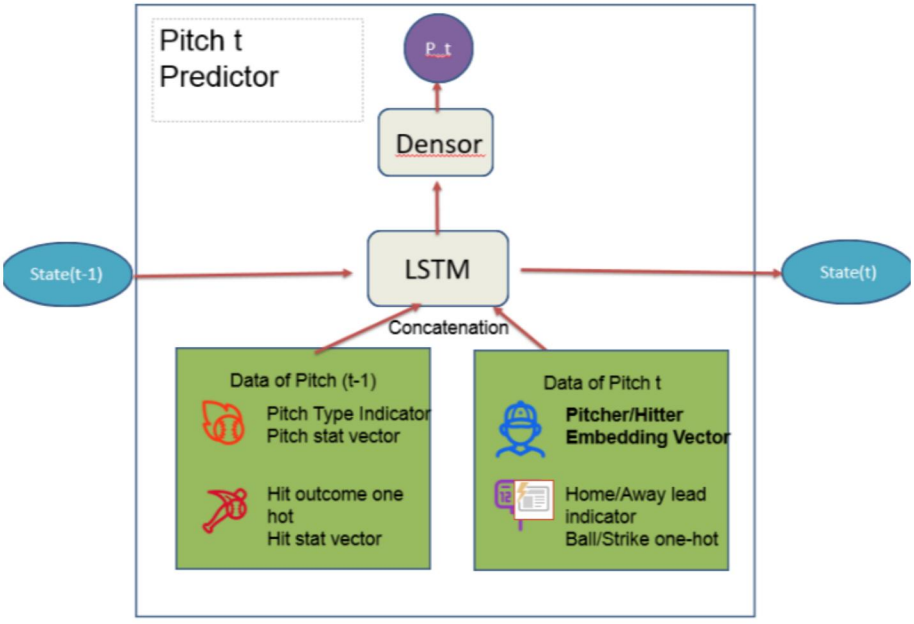

Figure 3: RNN Network Architecture to Predict the Next Pitch Type

Since there are thousands players in MLB and each player only provides limited amount of data points for a season, using one-hot vectors to represent players tends to make the model harder to train and less generalized. We try to learn a representation of each player. To set this up, we define a dummy learning problem by predicting:

$$\Pr[\text{pitch is thrown as a fastball}|\text{pitcher id, hitter id}]$$

We build a DNN model as Figure 4 to predict the probability, which takes one hot vector of a pitcher to a fully connected neural network with ReLu as activations between hidden layers, and a similar but another neural network of hitters. We call outputs of these two networks as embeddings of pitchers and hitters respectively. Finally, we concatenate these two vectors as a single one and with one fully connected layer and sigmoid activation as fastball probability prediction. We use the same dataset to train this model, but with only player ids as features, and pitch type as the label.

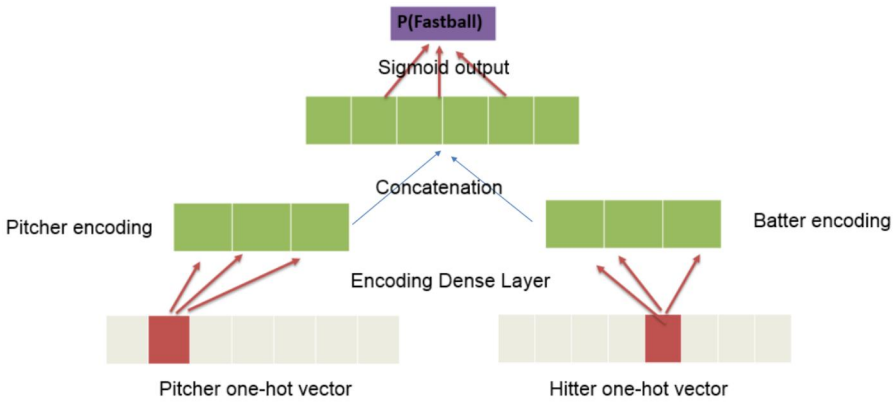

Figure 4: Player Embedding Network

The loss of both learning problems are defined as binary cross-entropy, i.e., for the pitch type prediction on a single game,

$$-\frac{1}{T}\sum_{t=1}^{T}\left[y_t \log(\hat{y}_t) + (1-y_t)\log(1-\hat{y}_t)\right]$$

where y_t is the indicator function that pitch t is a fastball, and $\hat{y}_t$ is the predicted probability. The accuracy is calculated as truncating the predicted probability at 0.5, i.e.,

$$\frac{1}{GT}\sum_{g}\sum_{t}\mathbb{1}\left[y_{g,t} = \mathbb{1}[\hat{y}_{g,t}) > 0.5]\right]$$

5 Experiment Results

We use the following hyper-parameter setting for our network: 3 hidden layers for the output of RNN, 32 as the size of the LSTM unit, 3 hidden layers to learn 32-dimensional player embeddings. For the optimization algorithm, we choose Adam with $(0.9, 0.99)$ and learning rate 0.001.

We obtain a training accuracy 73.3% and a test accuracy 66.2%. Note that the naive algorithm (predicting all pitches as non-fastball) gets an accuracy 62.8%.

6 Discussions

In this project, we models the baseball next pitch problem as a time series binary classification supervised learning task, and employs RNN to learn a model to predict that. However, we did not obtain results much better than the naive way or previous baseline [5] due to following reasons: limited architecture/hyper-parameters tried, test/dev data is not aligned well with training data (baseball players and games are changing from year to year), and data amount problem (not that many data in sports comparing to images/audio).

To improve this model in the future, we would like to try more neural network architectures and train the model by more hyper-parameter settings, consider other dummy problems for the player embedding learning, and split training/dev/test data in other ways. There are also some other topics

interesting to explore: find out what features are important for predicting the next pitch, how to simplify the model so that people can hand calculate (note that electronic device is not allowed in MLB game for players and coaches, but we can only need to predict for limited amount of players in each game), transfer the learning model to other baseball learning problems like player performances or game results.

References

[1] Baseball savant data scraping. `https://github.com/alanrkessler/savantscraper`.

[2] Baseballsavant. `http://baseballsavant.mlb.com`.

[3] Why it's almost impossible for fastballs to get any faster. `https://www.wired.com/story/why-its-almost-impossible-for-fastballs-to-get-any-faster/`.

[4] Michael A. Alcorn. (batter|pitcher)2vec: Statistic-free talent modeling with neural player embeddings. In *MIT Sloan Sports Analytics Conference*, 2018.

[5] Gartheeban Ganeshapillai and John Guttag. Predicting the next pitch. In *MIT Sloan Sports Analytics Conference*, 2012.

[6] Kaan Koseler and Matthew Stephan. Machine learning applications in baseball: A systematic literature review. *Applied Artificial Intelligence*, 31(9-10):745–763, 2017.

[7] Michael Lewis. *Moneyball: The art of winning an unfair game*. WW Norton & Company, 2004.