Strategizing in Baseball – Machine Learning and Applied Data Analytics

Kyle Benbrook, Justin Carter, Karthik Chinnappan,
Aaron Chumsky, Donald Falk, Andrew Klang, Jason Miller

Jeffrey Paul Wheeler (Faculty)

Department of Mathematics, the University of Pittsburgh

Pittsburgh, PA 15213, USA

jwheeler@pitt.edu

Abstract

As a baseball hitter, the ability to know what pitch is coming puts you at a huge advantage. In our project, techniques for predicting pitches are explained and implemented using a variety of methods including markov chains, random forests, and neural networks. By using our algorithm, we try to help put the Erie Evil Geniuses baseball team in a favorable position over their competition.

1 The Problem

1.1 Statement

We have been tasked by the Erie Evil Geniuses baseball team with providing specific advice for hitters against specific pitchers. With that being said, our objective is to identify patterns in opposing pitchers' games by predicting their pitches.

Ideally, we'd like to find information such as whether we can safely say that a pitcher stops throwing his curve ball when there are 3 balls, or if he avoids his fastball when facing a power hitter. Being able to answer questions such as these can give a hitter a major advantage when stepping up to the plate against any pitcher.

1.2 Data

The Erie Evil Geniuses provided us with a very large data set containing over 2.1 million observations. This data includes a wide range of categories including the pitcher name, batter name, pitch location, pitch type, pitch spin rate, pitch result, and play result. These data points are from games played in the 2016 through 2018 seasons.

We began noticing problems with this data set early on. First of all, many of the variables consisting of the play results had missing data points making this data difficult to use. Additionally, not many of the variables in our data set are particularly useful in predicting what pitch is being thrown. For example, the pitch speed and pitch spin rate don't help a hitter predict what pitch is coming because those are results that are only available after the pitch is already thrown. As a result, we needed to collect extra information to provide us more useful variables.

In our next step, we found hitter statistics on Fangraphs for more than 1,400 batters and merged this with our current data. These statistics covered many different tendencies of our hitters such as player performance statistics (batting average, Home Runs, etc), chase rate for pitches outside of the strike zone, where the player hits the ball, how often the player hits the ball hard, and much more. With this information, we hope to develop models that can make strong predictions for a pitcher's tendencies in a given situation.

1.3 Cleaning the Data

In addition to merging new data, we also applied data wrangling techniques to make our data-set easier to work with. First of all, we eliminated observations with missing variables so that our models could function properly. Additionally, we separated the data based on right and left handed hitters to create a

categorical variable for pitch location. Using this, we created boundaries in the strike zone using horizontal and vertical coordinates to establish categories for pitch location. Finally, we eliminated pitch types that were included in the data set, but not actually being thrown by any of the pitchers.

2 Initial Approaches

Given that this is a classification problem in which we are trying to predict pitch types, our variable of interest is categorical. Therefore, we're trying to minimize classification error, or more specifically, the number of pitches that are incorrectly predicted. Thus, our goal will be to minimize the following function:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

where y_i is the actual categorical response and $\hat{y}_i$ is the predicted categorical response. The indicator "I" will equal zero if the two response variables are equal and will equal one if they are not equal and thus are misclassified.

2.1 Markov Chains

2.1.1 What is a Markov Chain?

A Markov Chain is a stochastic process that seeks to describe a sequence of events using the probability of an event occurring, given a current state. Baseball can be seen as a discrete set of actions or events: Each pitch can be seen as a state that results in a finite number of outcomes. When we viewed baseball in this way, we realized that a Markov Chain could be used to predict the probability of what may occur next, given what had just happened. A Markov Chain visualization looks like this:

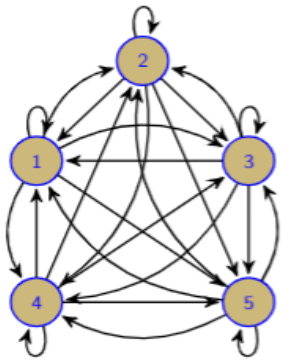

Figure 1: Markov Chain Depiction with 5 states

The Markov Property is given by:

$$P(X_n = i_n | X_{n-1} = i_{n-1}) = P(X_n = i_n | X_0 = i_0, X_1 = i_1, ..., X_{n-1} = i_{n-1})$$

2.1.2 Implementation

To implement the Markov Chains, we needed to use a Multi-State Markov and Hidden Markov Models in Continuous Time package. In order to get this to work, the data needed to be formatted in a particular way. We had to filter the data so that we were only working with pitches thrown by a single pitcher and we had to rearrange the data so that pitches were ordered sequentially based on the game they were thrown. We then created a transition matrix to model the possible states of the next pitch based on the different previous pitch states. This can be repeated to find probabilities of specific pitching sequences, helping users to determine strategies against a specific pitcher. Given an initial transition matrix, we can find the k-th step transition matrix, the likely hood of a certain outcome given k events occurring in sequence, by finding the product of previous transition matrices. This means that a transition matrix has the property that the product of previous transition matrices describes a transition of states along the given time-interval.

Having a Markov chain for a pitcher is helpful for a number of reasons. First and foremost, We can use the results of a pitcher's Markov chain to help understand how a pitcher transitions between his pitches. If we adjust our samples, we can create Markov chains that are based in certain situations like with a runner in scoring position, or a power hitter up. This would reveal a patterns for that pitcher in that situation. On top of that, we can also use a specific pitcher's Markov chain to replace missing pitch data, as the Markov chain could provide us with the pitch that was most likely thrown in a specific situation.

2.2 Tree Based Methods

2.2.1 Decision Trees

Throughout our research, we have experimented with classification trees as a tool for predicting the next pitch to be thrown by a pitcher. Typically, a tree is built or grown from select predictor variables – the splits or internal nodes – and the category for the desired outcome – the terminal nodes. In the case for a simple tree, we found it useful to classify our response, the type of pitch thrown by a pitcher, by variables including horizontal location, vertical location, the count and the outcome of the pitch (ball, strike, foul, etc). The algorithm of a classification tree creates a binary tree that finds the best variable to split using an appropriate impurity criterion known as the Gini criterion. At the end of these splits is a response value that is chosen based on a majority vote of the response variables in the terminal node of that split. The primary goal of each split is to reduce a measure of impurity between actual and predicted values:

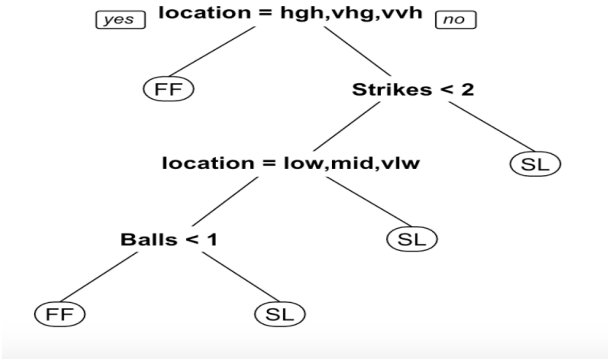

Figure 2: Categorical Decision Tree

$$Gini = \sum_{k=1}^{K} P_k(1 - P_k)$$

This equation is minimized when all cases at a terminal node for a split are in the same class. Therefore, a decision tree is making splits to the data in a way that minimizes the amount of data in a terminal node that is incorrectly classified.

2.2.2 Decision Tree Example

Figure 1 above is a simple example from our data where pitcher Chris Archer's pitches are being classified as either a fastball or a slider. The first split being made is the most important split for this classification problem. If the variable location is either hgh (high but still in the strike zone), vhg (a little above of the strike zone), and vvh (way above the strike zone), then the pitch is predicted to be a fastball (FF). If it's not at either of those three locations, another split is then considered. The first split is made because it will decrease the gini criterion more than any other split. In other words, for this split, the probability that the observations at the terminal node labeled "yes" will be correctly classified is at a maximum compared to all other possible splits. This same methodology is used to consider all subsequent splits.

2.2.3 Random Forest

The problem with decision trees is that it's only making predictions based on one tree. We need an algorithm that builds multiple trees and makes predictions based on results from multiple trees: This is where Random Forests come

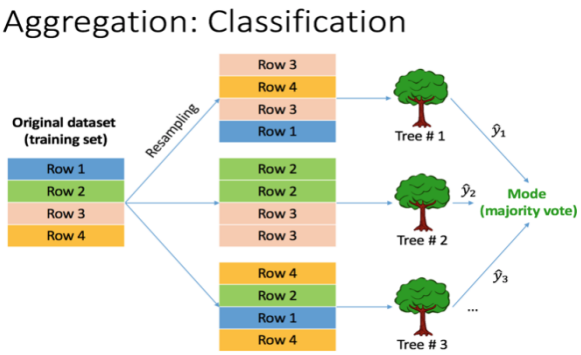

Figure 3: Random Forests

in. Random Forests utilize a method known as bootstrapping. The bootstrap method allows for resampling our dataset and thus being able to generate a collection of "different" datasets (variations of the same dataset that are being sampled with replacement). Additionally, in Random Forests, tree splits are made in a way in which only a subset of predictors are considered at each split. This way, not all trees are using the same variables to conduct each of their splits, which thus captures important relationships that may have initially gone unnoticed. Considering a subset of predictors at each split also helps to decorrelate the trees. To make our predictions, we use Out of Bag (OOB) error, which measures the prediction error for a given row using the rows that weren't included in the bootstrapped tree. After all trees are produced, the outcome with the majority vote given a set of input conditions will be picked in the final model. The process is illustrated in Figure 2 above.

For our problem of predicting pitch types, we included over 50 predictors. This included, but was not limited to, information on the hitters' performance, where the hitters hit the ball, how often they swing at pitches outside of the strike zone, and how hard they hit the ball. Using random forests on our data, we were nearly able to predict pitches with an impressive 65% accuracy; however, the model was heavily predicting fastballs. If we have a pitcher who predominantly throws two pitches such as fastballs and sliders, this issue isn't too noticeable; however, it becomes much more prevalent as the number of pitch types for a pitcher rises. This means that the misclassification rate for fastballs was really low but when the pitch was different, it was less likely that our model predicted it. Therefore, pitch types such as changeups, curveballs, and sliders had higher misclassification rates. The model was more just seeing fastball as "the safest bet."

It appears we need to implement a better method that can correctly identify pitch types that aren't thrown at as high of a rate.

3 Artifical Neural Networks

3.1 Structure

A neural network is fully-connected network of nodes intended to model the structure of biological brains. Layers of neurons are connected such that the outputs of each layer serve as inputs to the next layer, creating a feed-forward effect that starts at the input layer and ends in the output layer.

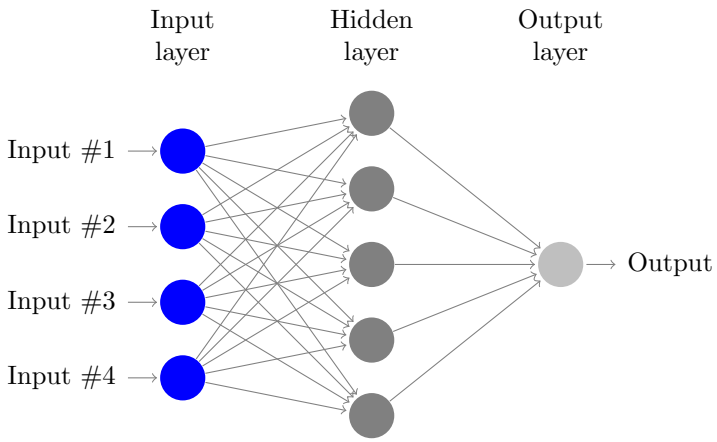

Figure 4: An example of an artificial neural network

Each neuron is defined by three components that are essential in the function of the overall network. These consists of the weights vector, biases vector, and activation function. Vector addition between the biases and the dot product of the inputs and weights amount to an intermediate value that represents that magnitude of the neuron's activation. This activation value is then normalized between 0 and 1 using the preferred activation function as network training is most effective when all values exist in this range. Common activation functions include the sigmoid function (shown below) as the various Rectified Linear Unit (ReLU) functions.

$$z = \sum \mathbf{w} \cdot x + b$$
$$\varsigma = \frac{1}{1 + e^{-z}}$$

The variables are defined as follows: $\mathbf{w}$ is the vector of weights from the preceding layer, x is the preceding activation, and b is the bias of the current neuron.

3.2 Learning

Once each layer has calculated its activation and the output signals have been defined, the model must then begin the process of evaluating the result and

tuning the internal network accordingly. Evaluation is done using a cost function which assigns a value to the undesirability of the output across a set of training data. A commonly used example of a cost function is the Mean Squared Error function:

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a||^2$$

The variables are defined in the following way: n is the size of the training set, $y(x)$ is the network output, and a is the expected result. Other functions such as forms of Cross-Entropy loss are also used by machine learning libraries.
The network then learns by making adjustments that minimize the cost function in a process know as backpropagation. The full backpropogation algorithm is as follows:

- Input: Set the corresponding activation a_1 for the input layer.
- Feedforward: For each $l = 2, 3, \ldots, L$ compute activation

$$z^l = w^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$

- Output error δ^L: Compute the vector

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- Backpropagate the error: For each $l = L-1, L-2, \ldots, 2$ compute

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- Output: The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$j and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Once the gradient of the cost function is determined, the weights and biases are then adjusted in the direction of the negative gradient. The magnitude of the adjustments and the frequency with which the backpropogation algorithm runs are controlled by the network implementer to further tune the learning process.

3.3 Implementation

We used the Keras machine learning library with a Tensorflow back end to implement the network. With this library, the implementation is relatively straightforward. The data is parsed from a .csv file into a numpy array. Next, the columns that represent the features and likewise the labels are defined. The next step is to one-hot encode the labels. We do this so that the classifications can be interpreted properly by the algorithm.

Now that the data is represented correctly, it is fed into the fitting algorithm. For this, we chose to implement a simple network of just one hidden layer with 12 neurons. The hidden layer utilizes ReLU activation while the output layer uses softmax activation. We used the ADAM optimizer to fit the network.

3.4 Results

Before trying to accomplish our end goal of being able to understand a pitcher's tendencies, we decided to implement a network that solves a simpler problem: Pitch classification. In other words, given quantitative data about the trajectory of a pitch (velocity, spin, break, position, etc.) can we train a network that can accurately determine what type of pitch it is? Using the implementation described above, we were able to fit a network that classifies pitches with 90% test accuracy. Satisfied with the results, we decided to continue to use this method to solve the problem.

To fit a network to the tendencies of a pitcher, we must be able to predict the next pitch that will be thrown. This is the same type of problem that we solved before, since both are categorical classification problems. The difficulty comes with the data that is to be used to fit the network. We decided to include the count(balls and strikes), the previous pitch type, the previous pitch location, and information about the batter to train the network. This process yielded a test accuracy of 46%.

4 Conclusion

With more data, specifically data that could be found on the scoreboard in the stadium (I.e. runs, the inning, runners on base, number of outs, etc) the performance of the predictive network could be greatly improved. It is not known rigorously that the inclusion of such data will improve the predictive performance of the network, but it is a heuristic based on the team's knowledge of the game. Thus, more data cleaning and experimentation is required.