

Maximizing Precision of Hit Predictions in Baseball

Jason Clavelli
clavelli@stanford.edu

Joel Gottsegen
joeligy@stanford.edu

December 13, 2013

Introduction

In recent years, there has been increasing interest in quantitative analysis of baseball. This field of study, known as sabermetrics, has been embraced by baseball managers as an effective means of predicting how players and teams will perform, and has gained mainstream popularity as a result of films like *Moneyball* and celebrity statisticians like Nate Silver.

In this project we apply sabermetrics, in the form of machine learning algorithms, to a difficult baseball prediction problem. We attempt to build a model that, given the set of all baseball players playing on a given day, chooses the player most likely to get a hit. In order to get a sense for the difficulty of the problem, we implemented a few trivial models. A model that chooses a player at random is correct approximately 60% of the time. A model that chooses a player randomly from the subset containing only the best hitters (with batting average greater than .330) is correct approximately 67% of the time. Our aim in this project is to find a more sophisticated model that increases this number as much as possible.

Data Collection

One positive aspect of working on a baseball prediction problem is that baseball data is plentiful and easily accessible¹. The first step of our data collection process was to write a Python script² that gathered all of the box score data from a given season into a single file. We then processed the raw box score data into a set of training examples, with a players state before a game as the input, and a binary indicator of whether or not that player had a hit in that game as output.

Feature Selection

Because of the easily quantifiable nature of baseball and the recent trendiness of sabermetrics, there are countless baseball statistics measuring everything from how many runs a player has produced (RP) to how many runs a pitcher allows in 9 innings (ERA) to the estimated contribution of a player to his teams wins (Win Shares). Since

¹All of our data was taken from <http://www.baseball-reference.com>

²A portion of our data scraper was copied from Benjamin Newhouse's SCPD Scraper, <https://github.com/newhouseb/SCPD-Scraper>

our problem focuses only on a player getting a hit and not on the game as a whole, we were able to eliminate the vast majority of metrics as possible features.

We considered as possible features all measurements that in some way quantified information about the batter, the pitcher, or the position of the fielders, as these are the factors that relate to the probability an at-bat results in a hit. By experimenting with different features, we determined that the following are good predictors:

- Average hits by the batter per game, last n games.
This measures how a batter has performed in the past. Average hits per game is more relevant to our problem than batting average, since it penalizes players for bunting, sacrifice flying, getting walked, hit by a pitch, etc.
- Average at-bats for the batter per game, last n games.
This measures how many opportunities the batter has to get a hit. This feature is extremely important; a mediocre hitter with 4 at-bats in a night is generally more likely to get a hit than an exceptional hitter with 2 at-bats.
- Average hits given up by the pitcher per game, last n games.
This measures the past performance of the pitcher. This feature was very predictive and, perhaps more importantly, independent of all of the batting features.
- Hitter-friendliness of the ballpark.
For this feature we used a statistic called

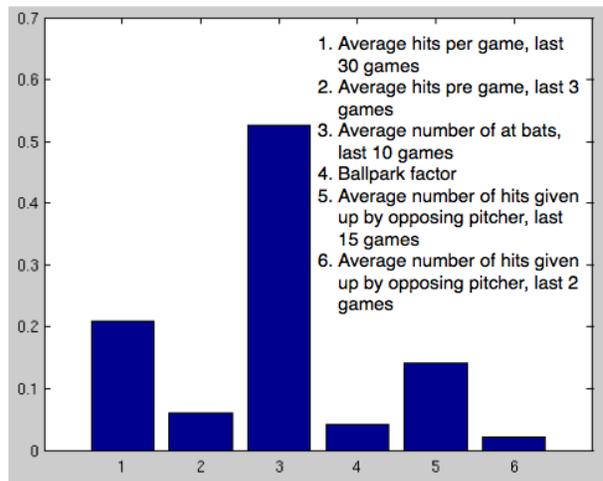


Figure 1: Relative weights assigned to each of the features by logistic regression

Ballpark Factor³, which approximates the advantage that a park gives to hitters, relative to other ballparks. A simple batting average over all of the at-bats in the park would not work, because that is highly influenced by the hitting ability of the home team. Using

$$\frac{\frac{1}{m} \sum_{i=1}^m \text{Total hits in home game } i}{\frac{1}{n} \sum_{j=1}^n \text{Total hits in away game } j}$$

(mostly) removes dependence on the home team by dividing by the team's performance in other ballparks.

For the measurements that consist of an average over the last n games, we created two features: one with large n , one with small. The large n feature represents how that player performs in general, while the small n feature measures whether the player is on a hot or cold streak going into a given

³<http://espn.go.com/mlb/stats/parkfactor>

game. As the figure 1 indicates, the general performance of a player is a better predictor than streakiness.

Performance Metrics

The first performance metric that we used was general error ($\frac{FP+FN}{FP+FN+TP+TN}$). However, since our data is unbalanced and our goal is more interested in avoiding false positives than false negatives, we quickly realized that precision is a better measure of our models success.

Another useful measurement looked at how well the model was able to separate positive and negative samples, using the difference between the average confidence assigned to the two classes. This was given by

$$a(1) - a(0)$$

where $a(k)$ is the average confidence assigned to samples with output k , given by

$$\frac{\sum_{i=1}^m \theta^T x^{(i)} 1\{y^{(i)} = k\}}{\sum_{i=1}^m 1\{y^{(i)} = k\}}$$

The difference in mean confidence is often quite small (see Figure 2), but improvements with this metric were strongly correlated with our final and most important metric, precision in the most confident subset (see Figure 3). This metric looked at the error rate over subset of the samples in which our model was most confident, which directly corresponds with the goal of this project.

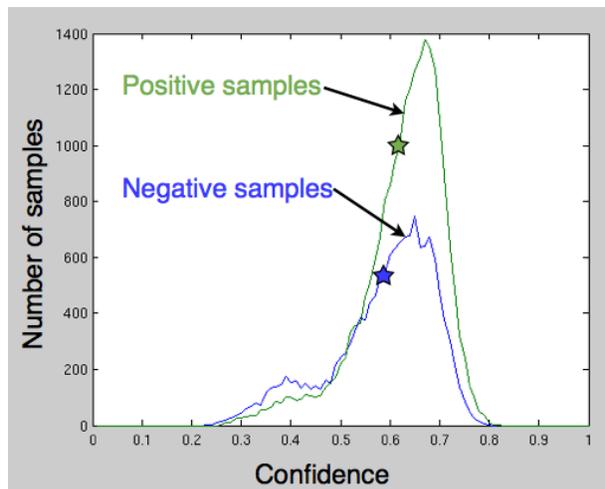


Figure 2: The distribution of confidence ($\theta^T x$), separated by positive and negative samples. The stars represent the mean confidence for positive and negative samples, respectively.

Logistic Regression

The primary model that we used was logistic regression. Logistic regression appealed to us for several reasons, the most important of which were:

- i) Simplicity of implementation.
- ii) As discussed earlier, logistic regression uses θ to assign a measure of confidence in each sample. This creates an easy way to choose a small subset of most confident samples.

After implementing a version of logistic regression that worked using any one feature, we ran into a major roadblock: our algorithm broke down when the training examples were higher dimensional. The gradient ascent took more than 50 passes over the design matrix

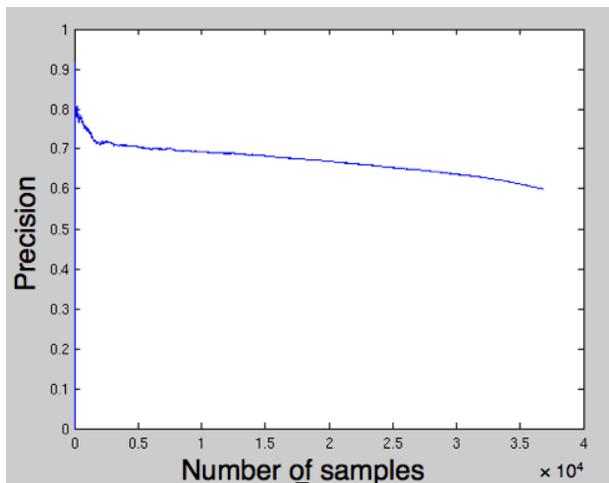


Figure 3: The precision over subsets of different sizes, where a subset of size n has the n samples in which the model is most confident. Note the sharp increase in precision for the subsets with fewer than 200 samples.

(which had 35000 samples) to converge and resulted in nonsensical values of theta. This issue resolved entirely once we normalized the design matrix; gradient ascent began converging within one pass over the data.

Our logistic regression results were encouraging:

Training Precision: 82.7%

Testing Precision: 79.3%

Training and testing precision are roughly the same, indicating that the model is not over fitting. However, we were never able to reach a training precision greater than 83% with logistic regression, indicating that the model has high bias and that the optimal decision boundary may be highly nonlinear.

SVM

Another model that we used was a support vector machine with a radial basis function kernel, which we hoped would address the apparent nonlinear decision boundary better than logistic regression did. Using the same features as with logistic regression, we were able to reach 96% training precision on a training set with 70,000 samples, which indicates that some form of separation is possible. However, this model generalized very poorly, with only 63% precision when it came to testing error.

Conclusions

Our results were very encouraging, overall. The testing precision achieved by logistic regression ($\sim 80\%$) was considerably better than random guessing ($\sim 60\%$) or trivial selection algorithms such as choosing the best hitters ($\sim 67\%$). Additionally, the fact that we were able to achieve very high training precision with the SVM indicates that, given the right tweaks to reduce over fitting, the SVM has the potential to outperform logistic regression.

As an alternative to logistic regression and SVM, we have begun experimenting with neural networks. Initial attempts have resulted in approximately the same testing precision achieved with logistic regression.

In addition to being able to predict hitters with reasonable accuracy, we also know which factors contribute the most to this prediction (See Figure 1 again). We can see

that the number of at bats is far more correlated with hits than anything else is, and that overall performance of a player is more correlated with hits than recent performance is, both of which are very non-intuitive results.

We are also planning to add additional, more nuanced features. Some of these include the handedness of the pitcher and batter (because the batter has a significant advantage when he and the pitcher have opposite dominant hands), the time of day that the game takes place, and the favored pitch types of the batter and the pitcher.

References

- SCPD Scraper, Benjamin Newhouse, <https://github.com/newhouseb/SCPD-Scraper>
- Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
- MLB Park Factors, <http://espn.go.com/mlb/stats/parkfactor>