Adi Srikanth
Fall 2020

MLB: Swing Prediction Analysis

Introduction

The goal of this analysis is to probabilistically predict whether or not a player will swing at a ball that comes their way. This analysis ultimately utilizes a Random Forest Algorithm in order to achieve this goal. Using 1500 estimators, this algorithm concludes with a 79.96% Accuracy Score and a 76.72% F1 Score. The model is expected to scale well and can handle both additional data records and additional features. The analysis that follows details the process and decisions made in order to produce the aforementioned model.

Note: this analysis is intended for an audience with some, but perhaps not a deep understanding of Statistics and Baseball. However, the analysis should generally be understandable to those without this background.

Setting Expectations

Of course, upon embarking on such an analysis, the most idealistic goal is perfect accuracy and prediction. However, given the inherent randomness of baseball, along with various shortcomings in the available data, it was important to establish a benchmark for performance. The most naïve model, which would choose *is_swing* as zero every single time, would evaluate to around a 53% accuracy mark. A model with almost no pre-processing, hyperparameter tuning, or other feature engineering appeared to evaluate somewhere in the mid-60% range for accuracy, depending on the model. Various other attempts available online, including one published by *Baseball Prospectus*, evaluate to an accuracy somewhere in the mid-70% range. Finally, an attempt published in *The Hardball Times* evaluated a model to predict contact vs no-contact on a swing to an 82.3% accuracy rate, which beat the naïve approach of always choosing no-contact by only a couple percentage points.

All of this to say that predicting baseball is certainly a challenge, and model-builders should contextualize their work and also be weary of certain results. (For example, given the state of prediction in baseball, a 99% accuracy rate likely has some issues lurking in the dark).

Adi Srikanth
Fall 2020

Data Used

The data used was acquired from an anonymous Major League Baseball Team. It is not specified the extent to which the data has been synthetically altered or created. However, the data appears to generally be reliable and applicable to general use cases.

The raw data columns provided is attached in the appendix as *Figure One.* There was no data dictionary provided, but most of the variables can be mapped to commonly used terms in sabermetrics. The *is_swing* column, used as the dependent variable, was generated using the *pitch_call* column. The data consisted of 1,077,997 rows to train and test on.

Data Cleaning and Pre-Processing

In order to bring the data to a format that could be fed into a Machine Learning Model, some prerequisite steps were required. These steps are outlined below.

The very first task at hand was the generate the dependent variable, *is_swing*, which was not provided in the dataset. Fortunately, this was reasonably straightforward as the *pitch_call* column outlined the result of the pitch. *Pitch_call* reported one of the following results for the pitch: BallCalled, BallIntentional, FoulBall, HitByPitch, InPlay, StrikeCalled, StrikeSwinging. Based on these possible outcomes, *is_swing* was computed using backwards reasoning.

The next task was to map the *tilt* variable to a usable format. *Tilt* refers to the rotation in the x-z plane of the pitch and is reported in hours and minutes. (Note that *Tilt* is markedly different from *Spin Direction*). Given that a larger number here is not more tilt, but rather a completely different kind of tilt, *Tilt* is treated as a categorical variable instead of a numeric variable. To specify this as a categorical variable, the variable, which was given in various different datetime format, was mapped such that each time mapped to exactly one value. For example, 01:00 and 1:00 both mapped to 01:00 for this variable.

Subsequently, the *balls* and *strikes* columns were combined to form a single column called *Count*. The reasoning behind this is that balls and strikes alone are not fully indicative of the situation at hand for a pitcher and batter. For example, two strikes with zero balls is quite different from two strikes with three balls. As such, *Count* was created as a categorical variable in order to capture the plate situation. The *balls* and *strikes* columns were both dropped to avoid multicollinearity. Various feature importance and model score tests confirm this train of thought.

Finally, null values were dropped from the data set. Given the large number of records and the lack of skew in the null values, this was considered a fairly safe decision.

Exploratory Data Analysis

In order to better understand the data, a few different baseline analyses were run. Given that a completely separate test data set was given, a quick analysis was run to confirm that the two datasets were consistent in their variables, variable values, and data types. A class balance test was run on the dependent variable that yielded a 46.6% to 53.4% split, with only a slight skew towards *is_swing* values of zero. An outlier analysis was also run and no variables showed any values above three quarters of a standard deviation above or below the mean value for that variable. As such, no outliers were removed.

A correlation matrix was also generated, but not visualized given the high dimensionality. There were several variables with high correlation, over 80% correlation, that required treatment and more pre-processing.

Otherwise, various histograms and distribution analysis revealed a fairly clean and non-worrisome dataset.

Feature Engineering

The first challenge approached regarding feature engineering was the handling of categorical variables. The two perhaps most popular methods of categorical variable encoding are One Hot Encoding and Target Encoding.

One Hot Encoding is the more traditional method of handling categorical variables. Essentially, for a column (let's call it *Letter*) with possible values A and B, columns isA and isB are created. If a record has the value B in the *Letter* Column, isA evaluates to 0 and isB evaluates to 1. This concept is extended if there is a possible value C, D, etc. The original column, *Letter* in this case, is typically dropped.

An example is show below:

Letter	isA	isB	isC	Dep_Variable
A	1	0	0	1
C	0	0	1	1

Target Encoding takes a slightly different approach. For each possible value of a variable, Target Encoding computes the average dependent variable result and replaces the variable value with the average. For example, say for all the records where the *Letter* column is C, the target variable is an average of 0.7. In this case, all the C values would be replaced with 0.7. The same would be done for all other values of *Letter* until the entire column becomes a numeric. Then, the column is treated as a numeric. Another example is shown below using the same data as above:

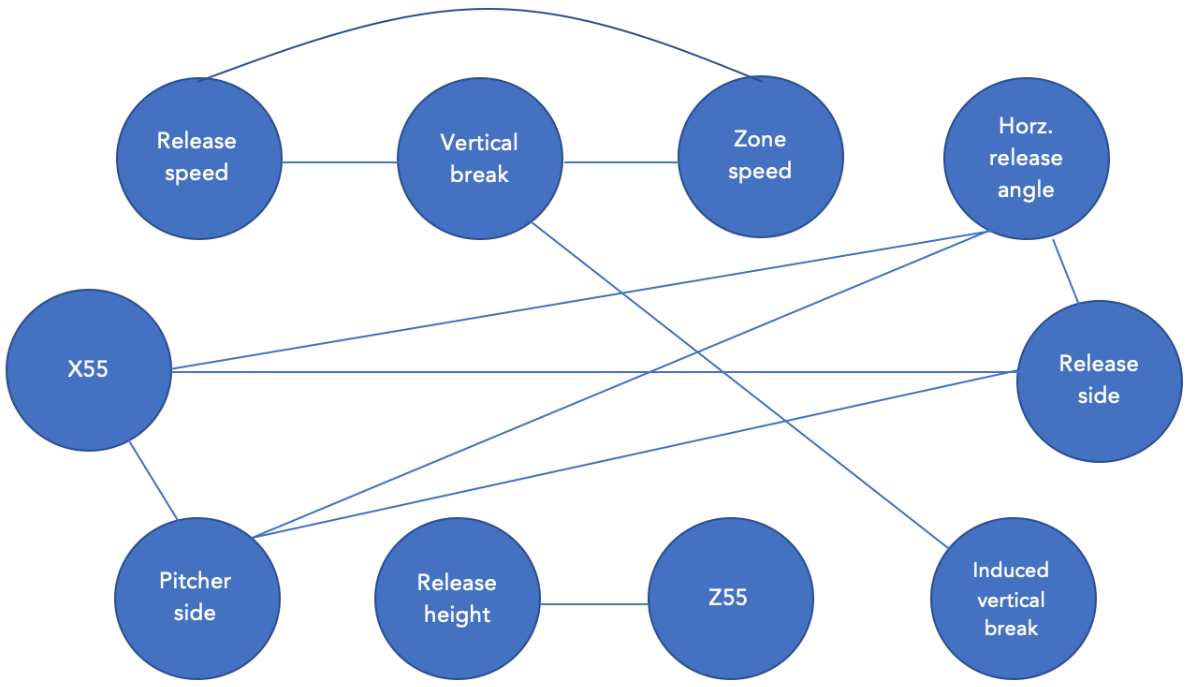
Letter	Dep_Variable
0.7	1
0.4	1

In this particular context, Target Encoded was chosen instead of One Hot Encoding. The reasoning that drove this decision was primarily to avoid high dimensionality. Multiple columns in the dataset had 20+ possible values and in One Hot Encoding, each of these possible values would become a brand-new column. Ultimately, One Hot Encoding would have increased dimensionality by more than a factor of 10.

At this point it should also be noted that the features in the data were not scaled using a Standard Scaler, logarithmic scaler, or anything of the sort. Given that Random Forest algorithms are fairly well-equipped at handling different value ranges for different variables, this seemed unnecessary. However, feature scaling is near impossible to cause significant harm to the performance of the model.

The next challenge was to deal with the high correlation variables mentioned in the Exploratory Data Analysis portion. Upon analyzing these variables, it became clear that there was not only correlation between sets of two variables, but also correlation between three or four variables. The graph below depicts the correlation between the variables identified as "high correlation variables" and uses the edges between nodes to define a correlation.

Adi Srikanth
Fall 2020



Given that there were pods of variables correlated that were related in definition, these variables were fused together using a process called Principal Component Analysis (PCA), which combines the direction and magnitude of related variables into a single variable in order to avoid multicollinearity.

Specifically, the variables *x55*, *horz_release_angle*, *rel_side, pitcher_side_Encoded* were combined to form *hor_location*, reflecting the combined information regarding the horizontal location of the pitch. Similarly, the variables *release_speed*, *vert_break*, *zone_speed*, *induced_vert_break* were combined to produce *ver_location* to encompass information regarding the vertical location of the pitch. The variable *Z55* was removed given that it was only correlated with *release_height* and had a significantly higher overall correlation score than *release_height*.

At this point, it is worth noting why PCA wasn't conducted on the entire dataset in order to reduce dimensionality. While PCA is an excellent way to reduce many variables into a few salient ones, PCA abstracts the meaning of a single variable. In the cases above, PCA was able to maintain some meaning of the variables, but only because the variables plugging into the PCA algorithm were well-defined and close in definition. Even so, some of the meaning of the new variables is abstract and difficult to interpret. PCA on the entire dataset may have improved accuracy, but at a severe cost of understanding the meaning of the algorithm's results.

Boruta Feature Extraction

At this point even after reducing some high dimensionality features, the dataset did seem to have a lot of features, with at least a couple of them appearing plainly dubious in terms of whether they would contribute to the intended model. Instead of manually pulling out variables that appeared unhelpful, the Boruta method of feature extraction was put to use. Boruta functions essentially by taking each variable in a data set, replicating it, shuffling the replicated column, and attaching it back to the dataset. For example, say we had a dataset like the one referenced earlier; Boruta would produce the following data:

Letter	Letter_Randomized	Dep_Variable
0.6	0.9	1
0.1	0.6	0
0.9	0.1	1

In the case of the actual dataset, a randomized column was generated for each variable. What Boruta does next is it takes all of the randomized columns and selects the randomized column that performed the best in terms of predicting the dependent variable. Then, Boruta says that if any of the original non-random columns was a worse indicator of the dependent variable than the best randomized column, that original non-random column gets tossed. This entire process is repeated many times to ensure statistical significance. The basic idea here is that no column in the data set is worse than simply having a random column of data – each column must have a non-random approach towards predicting the depending variable. Boruta ultimately tossed out 10 features, significantly reducing noise in the dataset and also reducing unimportant dimensionality.

At this point, the data was declared usable for a model.

Model Selection

Given the sheer number of variables still used in this model, the odds of a non-linear model being the best predictor was pretty high. Still, a logistic regression was run to confirm this thought.

Adi Srikanth
Fall 2020

After the logistic model was run, the candidate models were filtered to models that handle non-linear relationships between variables. At this point, three models were in contention: a Neural Network, a Support Vector Machine, and a Random Forest. There are many algorithms that could also have been implemented (notably, xgBoost) but were not chosen for various reasons ranging from industry success to implementation ease.

The Neural Network was quickly tossed out due to concerns over the interpretation of its results and how usable it would be for a Major League Baseball team. The thought process for tossing out the Neural Network was very similar to the thought process behind choosing not to use PCA on the entire dataset discussed earlier.

In a few trial runs, the SVM significantly lagged behind Random Forest and as such SVM was tossed out.

Random Forest Hyperparameter Tuning

There are various parameters that can be tweaked in order to optimize a Random Forest algorithm. Based on research and prior knowledge, the following parameters were chosen to tune: Number of Estimators, Maximum Depth of Tree, Minimum Samples Split, Minimum Samples Leaf, Maximum Features, and Warm Start. A discussion of these hyperparameters is not included, but various online tutorials walk through the meaning of each one.

The Python library Scikit Learn offers a couple of methods to implement hyperparameter tuning. Two popular methods are GridSearchCV and RandomizedSearchCV. GridSearchCV exhaustively searches all combinations of parameters and reports the best combination. Randomized Search randomly chooses a set number of parameter combinations and reports the best combination. For this model, RandomizedSearchCV was implemented given the large number of records in the model and the large number of estimators preferred. (GridSearchCV may have taken over 24 hours of constant runtime, but could also have improved the overall model accuracy).

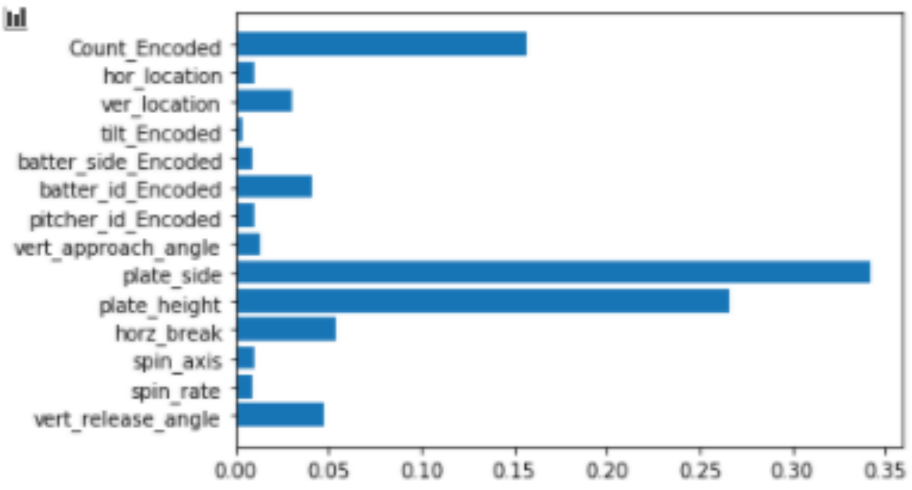
The parameters returned from RandomizedSearchCV were used after a cursory confirmation of their validity.

Adi Srikanth
Fall 2020

Model Performance

As noted in the start of this analysis, the model performed adequately, with an accuracy of nearly 80% and an F1 Score (which combines precision and recall) only slightly lower at 77%.

The model was designed with five folds for cross validation. The feature importance reported from the model is shown below, specifically valuing the variables *Count_Encoded*, *plate_side*, and *plate_height*; although, all the variables included in the model were deemed statistically significant beyond a random correlation.



Conclusion

Given that the industry discussion on this topic heavily uses accuracy as the metric of choice, this analysis will do this same. It is valuable to ask what the 20% gap between the model's accuracy and perfect accuracy is made of. It seems fair to say that around 5-10% of this gap is explained by the inherent randomness of baseball. Perhaps it is impossible to achieve perfect prediction, no matter how much data there is. In fact, it is likely this fact that makes baseball as exciting as it is.

Another 5-10% is likely explained by various deficiencies in the model. For example, GridSearchCV may have optimized parameters slightly more in order to produce a better model. Perhaps an xgBoost or Neural Network model also would be more effective in prediction. In cases like these, there are known trade-offs that balance accuracy with implementation and pragmatic business value.

Another 5-10% may be explained by the data at hand. Perhaps some data was thrown out too soon. Perhaps some data that would have been useful was never included in the original data set from the start. Or, the data could have been erroneous for a whole host of reasons.

The 20% gap is significant, but as outlined earlier in this analysis, the gap is not unreasonable. Improvements to hyperparameter tuning, data collection, and feature engineering (including feature extraction) can all improve this model. The problem at hand is certainly achievable to a pragmatic extent and baseball teams should look towards fine tuning this, if they have not already, in order to add this to their analytics ammunition.

Adi Srikanth
Fall 2020

Appendix

Figure 1

Data Field	Context
Date	Datetime
Level	Categorical, Denoting league (minors, majors)
Pitcher ID	Categorical
Pitcher Side	Categorical, Left or Right
Batter ID	Categorical
Batter Side	Categorical, Left or Right
Stadium ID	Categorical
Umpire ID	Categorical
Catcher ID	Categorical
Inning	Int
Top_Bottom	Int
Outs	Int
Balls	Int
Strikes	Int
Release Speed	Float, miles per hour
Vertical Release Angle	Float
Horizontal Release Angle	Float
Spin Rate	Float
Spin Axis	Float
Tilt	Time
Release Height	Float
Release Side	Float
Extension	Float
Vertical Break	Float
Induced Vertical Break	Float
Horizontal Break	Float
Plate Height	Float
Plate Side	Float
Zone Speed	Float
Vertical Approach Angle	Float
X55	Float

Adi Srikanth
Fall 2020

Y55	Float
Z55	Float
Pitch Type	Categorical
Pitch Call	Categorical
Pitch ID	Categorical
Is Swing	Categorical
Count	Categorical