Predicting Baseball Hall Of Fame Inductions

Michael Hirsch

ILLC, University of Amsterdam

michaelahirsch@gmail.com

Abstract

Every year, the Baseball Writers Association of America votes on a new Hall of Fame class. Each ballot consists of 10 votes, and players need to appear on 75% of ballots in order to be inducted into the hall. Players have 15 years to get inducted, and are no longer eligible if that time period has passed. Here we attempt to classify hall of fame batters based on their career statistics using decision trees and an ensemble method, random forests.

1. Introduction

Major League Baseball (MLB) has been keeping thorough records of batting, picthing, and fielding statistics since its innagural season in 1869. Recently, with the advent of sabermetrics by the Society for American Baseball Research (SABR), many new metrics building on traditional statistics were created. Such metrics caught the eye of stasticians, as these new metrics allowed for the creation of even more powerful predictive models.

In this paper, we are investigating what makes a Hall Of Fame (HOF) batter. There have been over 20,000 Major League baseball players in the history of the organization, and only 211 of them have been inducted into the HOF. There are several ways in which a player can be inducted, and we only concern ourselves with players inducted from Baseball Writers Association of America (BBWAA) ballots, as this process is regulated. Often, fans and players feel that a worthy candidate is unfairly denied entry into the HOF because of voter bias, stacked ballots, or negative associations with performance enhancing drugs. The model proposed in this paper should be able account for these cases.

We will investigate predictions based on both classification trees and random forests. Random forests, introduced by Leo Breiman of UC Berkeley

in 2001, is an ensemble learning technique that can be used for both classification and regression and improves on decision trees by correcting for overfitting. A random forest for classification consists of a collection of decision trees and outputs a prediction representing the most commonly occuring prediction of the decision trees.

2. Data

Data was collected from the Lahman Baseball Database, a freely available database that has been contiously updated since 1994 with the help of SABR and many individual researchers. There are many data tables available for download, but the ones that we are focusing on are *Batting.dat* and *HallOfFame.dat*. Batting statistics were supplied on a by-year basis, so a player's career statistics were computed by aggregating his yearly results. The batters' data was then merged with hall of fame voting results using the player's ID string. We will only be considering players who have appeared on at least one ballot, since failing to do so would indicate that that player is not hall worthy. All data preparation was done within R, the environment where we also build our learning models.

Attention is paid to 12 different batting statistics over the course of the player's career:

1. G: games played
2. AB: at-bats
3. R: runs
4. H: hits
5. X2B: doubles
6. X3B: triples
7. HR: homeruns
8. RBI: runs batted in
9. SB: stolen bases
10. BB: walks
11. SO: strike outs
12. Inducted: classifier for HOF induction

We needed to further filter our data by not including pitchers in our set of observations. This was done by removing those observations that indicated that the player recorded less than 300 RBI in his career. This number was

not arbitrary, but rather chosen by cross referencing my data with HOF data from Baseball-Reference.

Our software of choice for this paper is R. We will be using several packages for our learning techniques, indicated in the sections below. Before training our model, it is important to have a quick look at our data.

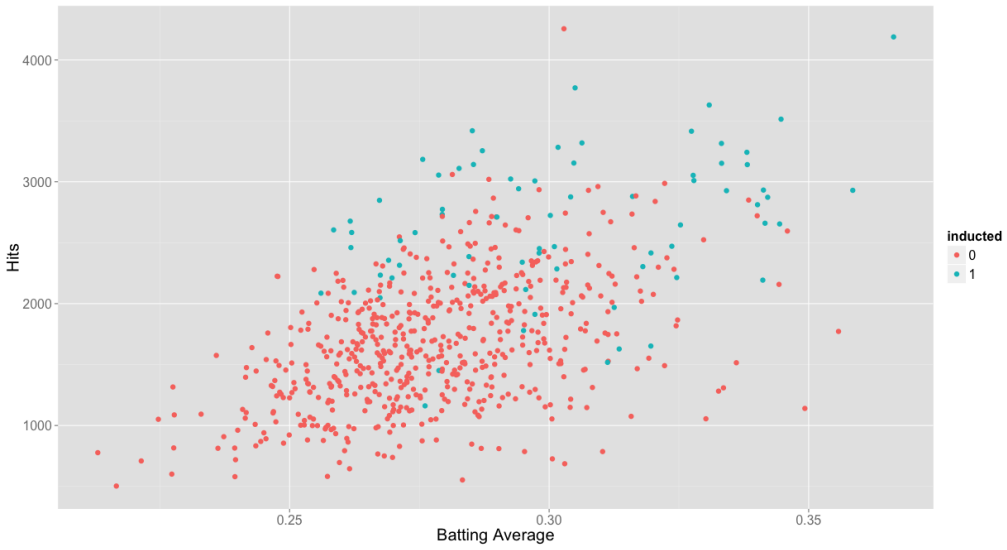

Figure 1: Batters with more hits and a higher batting average are inducted

We can see that more hits and a higher batting average seem to correlate with induction. This makes sense, as one often judges the quality of a batter by his batting average, the numbers of hits a player has per at-bat.

insert some more figures here

3. Tree-Based Methods for Classification

In this section, we will give an account of two learning methods that we will use for the basis of prediction: classification trees, and an ensemble method, random forests. We will also introduce the concept of bootstrap aggregating (bagging) which is a fundamental ingredient of random forests. The information here is theoretical, and we will encounter applications of these ideas to our HOF data in the next section.

3.1. Classification Trees

Classification trees are a fairly effective predictive tool that are lauded for their high degree of interpretability. When constructing a classification tree, we begin with the full set of observations and begin dividing the predictor space into non-overlapping regions by means of binary splitting. In prediction, we assign each observation in a given region of the predictor space to the most commonly occuring class of the training observations in that region.

When choosing splits in our tree, we are concerned with the *Gini Impurity* at each split. We want to choose the feature to split on based on which feature split will give us the most pure sucessive node. Gini Impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Gini Impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero)when all cases in the node fall into a single target category. This defined as:

$$G = \sum_{i=1}^{m} p_i(1 - p_i) = 1 - \sum_{i=1}^{m} p_i^2$$

where i takes on values in $\{1, 2, \ldots, m\}$ and p_i is the fraction of items labled with value i in the node. So G can be seen as a measure of total variance across the m classes. Since this paper will only be discussing trees with binary splits, m will also be taken to be 2. Note that $G = 0$ when all observations in that node fall into one class. In splitting at nodes, we look to minimize this value.

Let us illustrate this with a very simple example of a decision tree created to predict the survivors of a boat disaster. The root node contains all passengers of the ship. We then split the full set of observations according to their sex, followed by a split on what type of ticket they had.

Now, we can consider the Gini Impurity associated with the node labeled "male" in the decision tree. The table below shows the fates of all males based on their ticket type.
If we split this dataset on ticket type, we can calculate Gini Impurity for each of the two possible responses as follows, with $i = 0$ indicating survival and $i = 1$ indicating death:

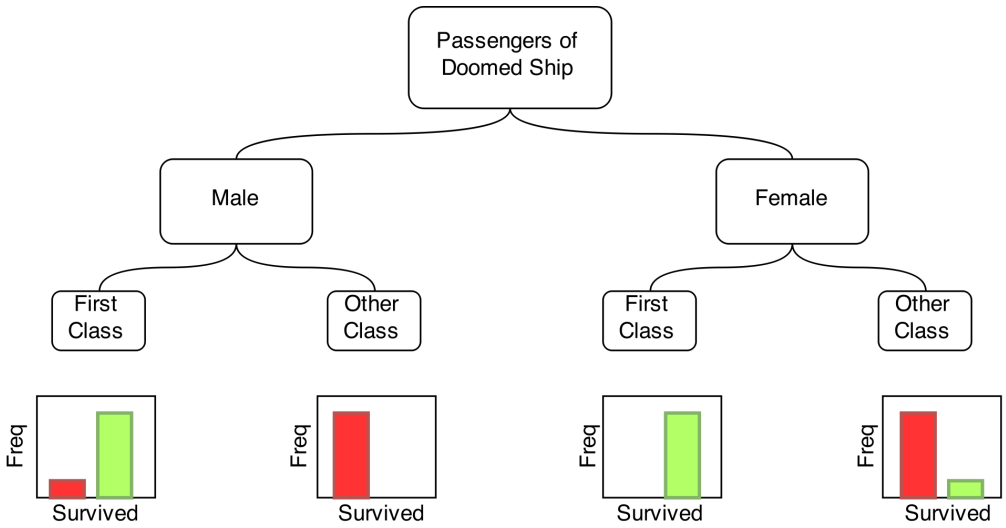

Figure 2: Decision Tree for Survival of Boat Disaster

	Survived	Died	total
First Class	122	19	141
Other Classes	12	1113	1125
total	134	1132	1286

Table 1: Aftermath of Boat Disaster

$$G(firstclass) = 1 - \sum_{i=0}^{1} p_i^2 = 1 - ((122/141)^2 + (19/141)^2) \approx 0.233$$

$$G(otherclass) = 1 - \sum_{i=0}^{1} p_i^2 = 1 - ((12/1125)^2 + (1113/1125)^2) \approx 0.021$$

This tells us that knowing that a male was holding a ticket other than first class leads us to a very pure node, that is, one in which we have a substantal amount of information to determine the fate of that passenger. Gini Impurity is related the related notion of entropy and can be seen as measures of how informative the answer to a "yes/no" question is at the split in a tree. Entropy can also be used in determining splits, and can calculated using the

following equation, where p_i is the same quantity as defined above:

$$-\sum_{i=0}^{m} p_i log_2 p_i$$

Decision trees, while useful, have their limitations. Firstly, it is possible to grow overly complex trees that overfit the training data [1]. In such cases, a method known as pruning is required to combat this problem. Cost complexity pruning is a method for considering subtrees of our initial tree modeal that minimize the *cost-complexity criterion*, which is defined as follows[2]:

Let T_0 be our unpruned decision tree, and define a subtree $T \subset T_0$ to be any tree that can be obtained by collapsing any number of internal nodes. Let $|T|$ denote the number of terminal nodes of our subtree , and let m be an index for them. Define the number of observations at each terminal node as:

$$N_m = |\{x_i \in R_m\}|$$

And the cost-complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m G_m(T) + \alpha|T|$$

For each α, there is a unique smallest subtree $T_\alpha \subseteq T_0$ that minimizes $C_\alpha(T)$ An estimation $\hat{\alpha}$ is done by using tenfold cross-validation, and the final pruned tree is $T_{\hat{\alpha}}$. In tenfold cross validation, the original training set is paritioned into ten equal subsamples, nine of which are used to train are used to train the model, and one used for validation. This process is repeated 10 ten, each time selected a different subsample as the test data. The 10 results from are then averaged to produce a single estimation. More details can be found in [3].

3.2. Ensemble Methods

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the individual learning algorithms composing the ensemble. Here, we look at two such methods, bagging and random forests.

3.2.1. Tree Bagging

Decision trees as discussed in the previous section often suffer from high variance, meaning that if we fit a decision tree to two random samples from a set of training observations, we could end up with vastly different results. Bagging is a method for reducing the variance of any learning method by taking repeated samples from a single training set and taking, in the case of classification, the most commonly occurring class among the boostrapped predictions.

On average, each of the B bootstrapped samples will take about two-thirds of the training observations. The remaining observations that were not used in fitting a bagged tree are referred to as the *out-of-bag* (OOB) observations, and we can predict the target varialbe for the ith observation using each of the trees in which that observation was OOB. This will yield, on average, $B/3$ predictions from which we can predict the classification for the observation by taking majority vote.[1] Using this method, we can get a single OOB prediction for each of our observations, obtaining an OOB error, which is a valid estimate of the test error for our bagged model.

Bagging also gives us insight into which of our predictors were the most important in creating our model by looking at the Gini Impurity. To do so, we take the sum of the amount that Gini Impurity has dereased by splitting on a given predictor and take the average over our B trees.

3.2.2. Random Forest

The random forest technique is quite similar to taking bagged samples of trees, with the main difference being that at each of the splits only a random sample m of the set of all predictors p is considered. Typically, $m \approx \sqrt{p}$ predictors suffice. This technique helps to decorrelate the tree, as there may be one predictor that has a very high influence in classification. In the case of bagging, this predictor will most likely be used as the first split in a majority of the trees, causing most bagged trees to be quite similar and correlated. By forcing each split to consider on a subset of the predictors, on average $(p - m)/p$ of the splits will not consider this strong predictor.

In this paper, we will be utiliziing the *randomForest* package in R.

4. Prediction Models

In this section, we apply the techniques introduced in the last section to our HOF data. We will start by creating a single decision tree based on some

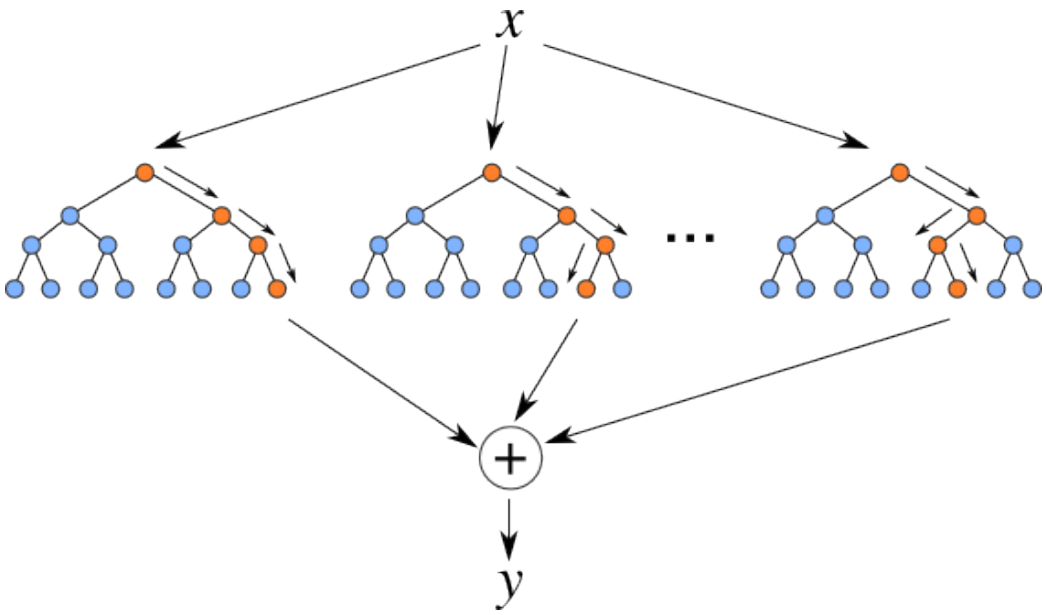

Figure 3: A Random Forest. The prediction for **y** is made by majority vote. **get a new figure**

training data, and then predicting the induction classifier of our training data based on our model. We will also look at some metrics that will allow us to test the effectiveness of our classification models in order to see what parameters would be best to use.

After evaluating our single decision tree, we will move onto random forest classification, to see how it improves our predictions.

4.1. Classification Tree

Using R's tree package, we can fit a classification tree to our data. To do so, we will split the entire data set into a training and a test set and build our tree using the training data. As our training set, we will take a random sample of about 2/3 of our data. This amounts to ≈ 400 observations. The following results were obtained by predicting induction based on all of our features except for playerID. The tree determined that runs, stolen bases, strikeouts, walks, home runs, grounded into double plays, at-bats, and hit by pitches were the most important features. We also see that the tree has a misclassification rate of 3.25% A quick look at the tree indicates that the number of runs a player has scored in his career is the most important

predictor, since the first split was made usings runs. This is not surpising, as runs are one of the only statistics that we considered that relate directly to a player's team winning a game.

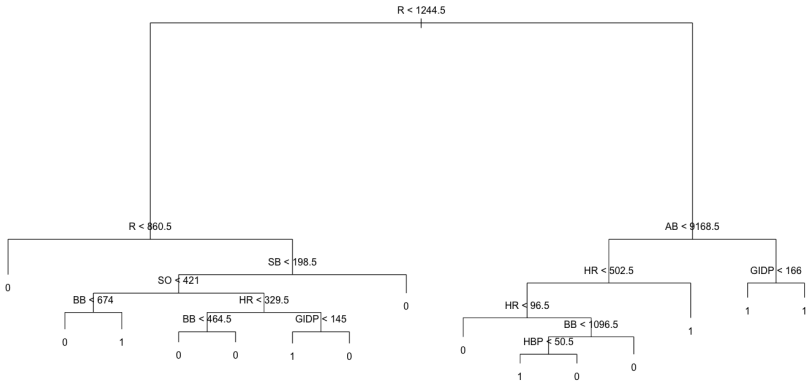

Figure 4: An unpruned tree

```
Classification tree:
tree(inducted ~ . - playerID, data = HallOfFame, subset = train)
Variables actually used in tree construction:
[1] "R"   "SB"  "SO"  "BB"  "HR"  "GIDP" "AB"  "HBP"
Number of terminal nodes: 15
Misclassification error rate: 0.0325 = 13 / 400
```

Now, to test the performance of this tree, we must estimate a test error by predicting a player's hall of fame induction on our test data. We will look at how our classification model performed.

```
tree.pred=predict(tree.hof,HallOfFame[-train,],type="class")
with(HallOfFame[-train,],table(tree.pred,inducted))
         inducted
tree.pred  0   1
        0 203  11
        1  19  18
```

This tells us that we have a misclassification rate of $1-(203+18)/(203+18+11+19) \approx 12\%$. We will see if this can be improved by using the tree

package's pruning feature. The goal of pruning a decision tree is to reduce the complexity of our model, and to reduce overfitting. A pruned tree is simply a less complex tree chosen from set of all subtrees of our decision tree. The method used by the tree package in R is *cost complexity pruning*, which operates as follows: Rather than considering every subtree of our initial tree, we only consider a sequence of trees that is indexed by a tuning parameter α. The goal of pruning is to select a subtree that will lead to the lowest misclassification error rate. Such an α can be chosen by using cross-validation on our initial tree.**talk about cv and pruning more?** We can see that the optimal size for a tree is between 4 and 7 terminal nodes. With this imformation, we can prune back our original tree to have a more interpretable and accurate model.

```
Classification tree:
snip.tree(tree = tree.hof, nodes = c(7L, 2L))
Variables actually used in tree construction:
[1] "R"  "AB" "HR" "BB" "G"
Number of terminal nodes: 7
Misclassification error rate: 0.04 = 16 / 400

         inducted
tree.pred   0   1
        0 207  10
        1  15  19
```

4.2. Random Forest

5. Conclusions

[1] Hastie, James, Tibshirani, Witten, An Introduction to Statistical Learning, Springer, 1st edition, 2014.

[2] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, CA, 1984.

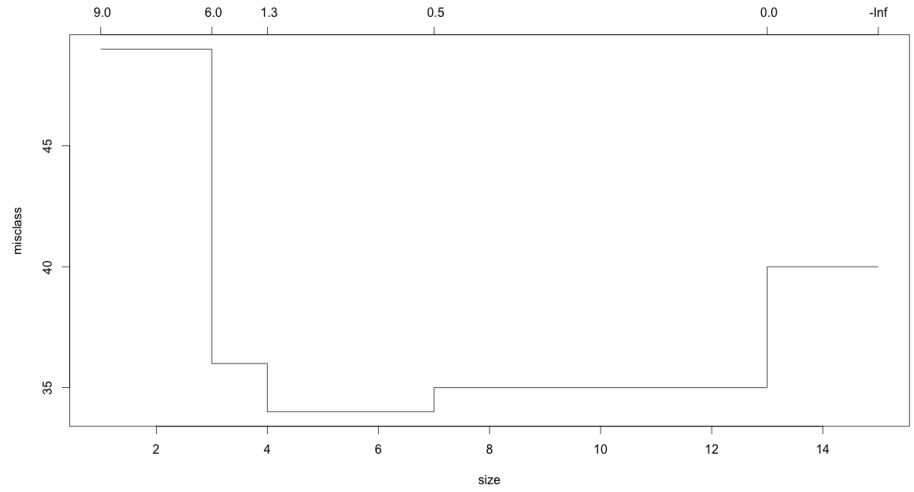

Figure 5: Tree size against misclassifications

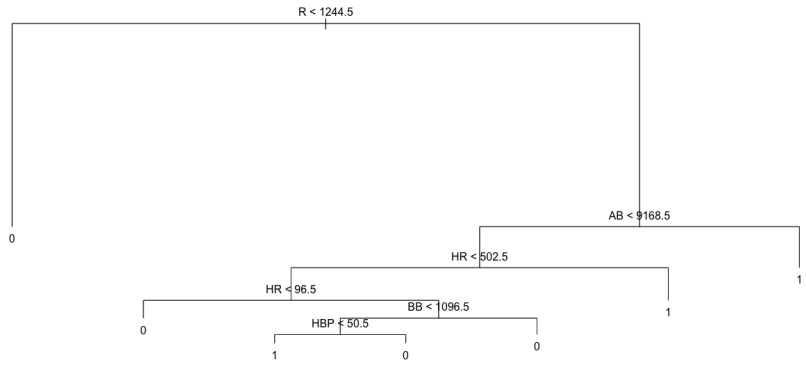

Figure 6: Our pruned tree with 7 terminal nodes