DS 740 Final Project

5 May 2021

Will a Major League Baseball Player Hit a Home Run?

Every night during the baseball season, every baseball manager faces a question: which players should I put in my lineup? The lineup – the players who will start the game – should obviously maximize the team's chance of winning the game, and one big part of that is maximizing the team's chances of scoring runs. In baseball—a notoriously random sport in which the placement of a rolling ball three inches to the left or right can mean the difference between winning and losing—the most certain way to score runs is via the home run. Every home run always leads to at least one run scored, which cannot be said for any other event in baseball. Moreover, home runs, in contrast to outcomes like singles or doubles, are considered to be one of the more predictable, less random outcomes in the game. Because home runs leave the field of play, they cannot be caught by the opposition: there is "no defending the home run," so there are fewer extrinsic variables to consider.

So: how could a manager seeking to maximize their offense set their lineup for a game based on data-driven predictions of how many home runs the team's hitters would hit? This project attempted to build statistical models that would identify players who would hit home runs in a given game.

I started from publicly available data from the Retrosheet database, which meticulously tracks every play in every Major League Baseball game. From the Retrosheet data, I was able to identify which players have and have not hit home runs in every game started by those players since 2006. Players who did not start in a particular game were excluded from the analysis because, by virtue of entering the game late, they almost always get fewer chances to hit home runs.

What data about player and game might affect whether we predict that a hitter is going to hit a home run? A vast array of data, ranging from the player's past performance, to the pitcher's past performance, to their physical characteristics, the park (some are smaller or have thinner air than others), the temperature, the wind speed and direction, perhaps even the time of day (are players tired if they have to wake up early for a game?). Combining the Retrosheet data with another database, Lahman's Baseball Database, I was able to consider the following variables for every combination of game and hitter:

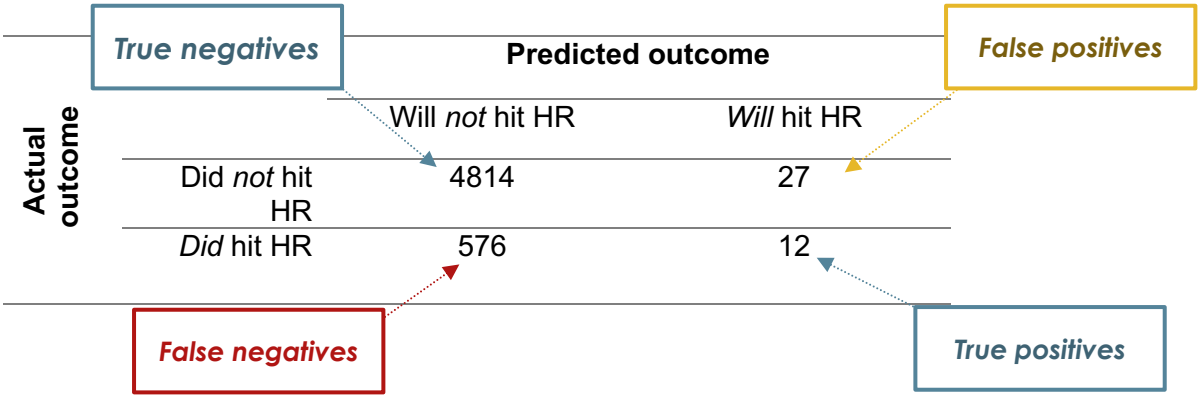
- The site of the game (the park)
- The month of the game
- The start time of the game
- Whether the game was at night or in the day
- The temperature at the start of the game
- The wind direction (for example, "out to right") and wind speed
- Whether the hitter was at home or on the road
- The player's position (pitchers were removed from the data)
- The hitter's weight, height, and batting handedness (whether they hit right-handed or left-handed)
- The pitcher's weight, height, and handedness
- Simple data about the hitter's performance in the season prior to the season the game was played, such as the number of at-bats, run scored, hits, and home runs the player had

The statistical modeling techniques that I used to try to predict whether the hitter would a home run in the game are known as random forests and neural networks. The random forests model builds a large number of "decision trees" that are posited to answer the question we are asking of the data, then calculates an answer to the question based on the aggregate results of these decision trees for a given piece of data. For example, one of the decision trees might posit that if the game is in Colorado and the hitter hit more than 20 home runs in the previous year, then the player will hit a home run. Another decision tree might posit that if the game time temperature is under 60 degrees Fahrenheit and the game starts after 7 pm, then the player will not hit a home run. Dozens of these trees essentially vote on the answer, and the algorithm tallies the votes and declares a winner.

Neural nets work similarly, but they assign a weight to various variables and pass them through a data structure that is designed to work like neurons in the brain. Similarly, votes are tallied, and a winner is declared.

Now, before we get to our answers, we must consider that home runs are uncommon events. In the data set I built from Retrosheet, just 11% or so of non-pitcher hitters hit a home run in any given game they started. So a smart algorithm is going to err on the side of guessing that a hitter is *not* going to hit a home run. And that was what these algorithms did: lacking any clear predictor that could really tell them that a hitter was going to hit a home run, they bet that the hitter wouldn't most of the time.

The best way to understand this is to look at what is called a *confusion matrix,* which is a simple table with two rows and two columns that tracks how often the model made the right and wrong predictions. The confusion matrix for the best statistical model developed from this data is as follows:



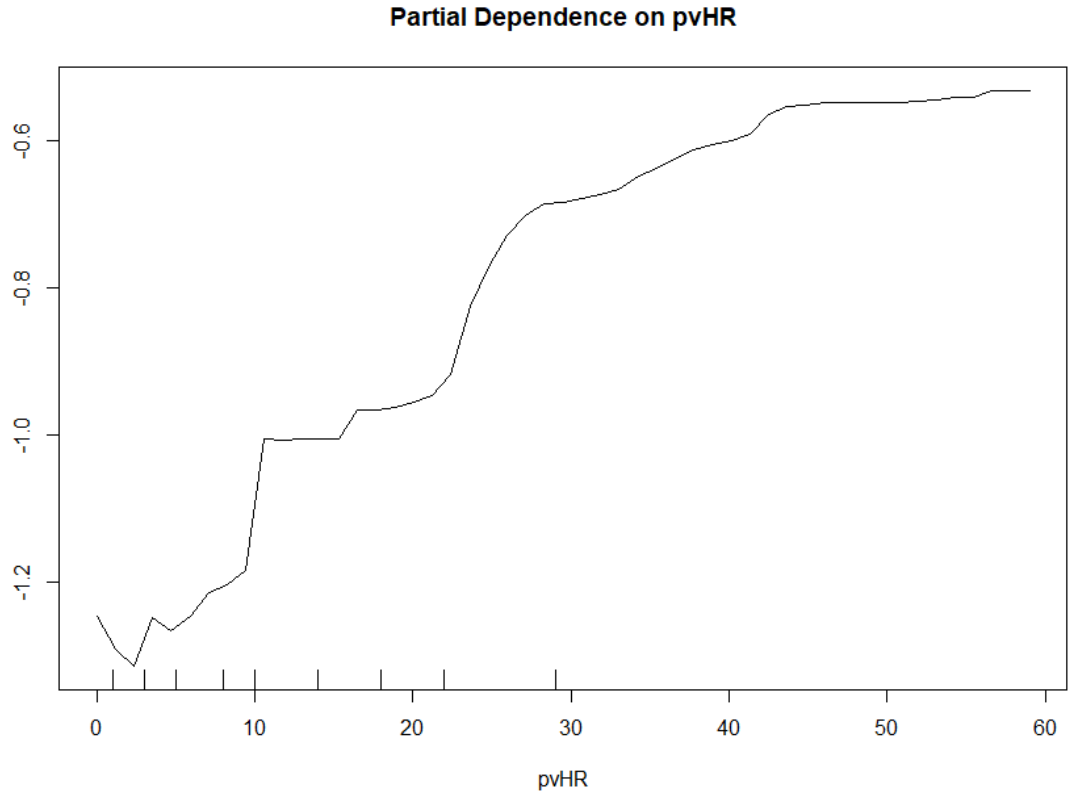
What this shows us is that, first, this model was very reticent to predict that a player would hit a home run. Among the over 5000 predictions it made, it only predicted that player would hit a home run 39 times. And we can understand why: most of its predictions were wrong. They were "false positives": instances in which the model predicted a home run, and the player did not in fact hit a home run. The model only made 12 "true positive" predictions: predictions of a home run in which the player did hit a home run. On the other side of the equation, the model missed 576 opportunities to correctly call a home run. These were its "false negatives": instances in which the model predicted no home run, and the player did hit a home run. And this was our *best* model: it was tied for the most true positives of any model.

Why do I rate the model with the most true positives as the best model? Because if you were a manager of a baseball team, you would well understand that it is very hard to know with certainty that any player will hit a home run in any given game. You routinely put players into the lineup in the hope that they might hit a home run, and if you happen to get one or two of these predictions right, you are happy. It is just a hard thing to predict. So you would happily choose a model with some "false positives," if it could count on a certain number of "true positives" from the model. In other words, this is a case of a model where you want to maximize true positives, and where false positives are not that bad – and false negatives are very bad, because they represent missed run-scoring opportunities.

If we wanted to improve this model, we would seek ways to get more positives out of it (ideally true positives, but we'd be fine with many more false positives). My hunch is that we could do that by adding more helpful predictor variables. The best model got its most useful information out of the following variables:

1. The hitter's home runs in the previous season
2. The hitter's RBI (runs batted in) in the previous season
3. The hitter's games played in the previous season

The effect of previous-season home runs on the predictions can be seen in the following plot, which shows that (unsurprisingly) more home runs hit by the hitter in the previous year mean an increased likelihood that the model will predict a home run – or perhaps more precisely, *not not* predict a home run (which is what the negative values on the left are telling us).



In other words, past performance was the best indicator of future returns for this model. Games played, the third-most-important variable, could be viewed as a stand-in for how valuable the player's manager

views him as being: the manager puts him in the lineup often, presumably, because he is seen as a good bet. Perhaps it would help to add more data about past performance: how the hitter was hitting as a rate (like home runs per time at bat) in the last 10, 50, or 250 at-bats, the hitter's recent strike-out rates, and so on.

Moreover, this model lacks much information about a key factor in baseball, the pitcher. Adding data about the pitcher's past performance would very likely add value. A hitter's performance against pitchers similar to the starting pitcher of a given game could assist in prediction.

In conclusion, what does this exercise tell us? First, that we are probably missing the right data, and second, that data about the hitter's past performance is worth exploring further. Baseball is a hard-to-predict sport, especially on the horizon of one game, and even high-tech data-mining techniques have trouble making very reliable predictions without access to the right data. These algorithms suffered from a lack of data. Luckily, there is a wealth of data available about baseball, including new data about a hitter's average exit velocity on hits and launch angles, and more can be added to the dataset.

It should also be mentioned that the models selected were for this project were hard to run with large amounts of data. They make a gigantic amount of calculations and are therefore time-consuming on a typical laptop. To run them in the amount of time available for the project, the author had to take a sampling of about 1% of the available data. Simpler modeling techniques should be explored.