MIT Sloan School of Management
15.285: Sports Strategy and Analytics, Spring 2021

A Simple and Efficient Algorithm for Recommending MLB Batters to Sit on Particular Pitch Types

Tyler Kim
Sawyer Koetters
Adam Katz
Matthew Nay
Graham Cartwright

May 2021

Contents

1 Motivation

In the game of baseball, one of the most fundamental skills is the ability to decide whether or not to swing on a pitch. Many factors make this decision incredibly difficult. In particular, pitchers often switch through four or five different pitch types (fastball, changeup, curveball, slider, etc.), making it incredibly hard for a batter to read the ball the moment it leaves the pitchers hand. Indeed, it's hard enough to predict whether a ball is going to be in the strike zone or not, but it's even harder to make that decision when the pitch can be varying in speed, horizontal movement, and vertical movement. If coaches or general managers were able to develop a framework with which to help players in this decision process, batters could potentially be exponentially more productive.

One can imagine a world in which the batters of a team knew exactly which pitch was coming.[1] This would be an incredible advantage since batters would be able to predict whether a pitch was going to be in the strike zone far more accurately than other batters. However, knowing this information is essentially impossible assuming a team does not cheat. Thus, since we don't have this luxury, we attempt to develop of strategy that could have a similar, albeit smaller, positive effect.

This is accomplished by developing a function that recommends a pitch type for a batter to sit on. We defining *sitting on a pitch type* to be only attempting to hit the ball if the pitch is the specified pitch type. For example, if we instruct a batter to sit on a sinker, the batter would only swing at the ball if the pitch is a sinker. By sitting on particular pitches, a batter may be able to maximize his expected productivity by strategically swinging on pitches that are his strong-suit and the pitchers weakness. This paper outlines the data and algorithmic design behind such a framework.

2 The Data

One of the first problems to tackle when thinking about this algorithm is what data will be needed. On a very rudimentary level, it is clear that we will need four "categories" of data in order to make a good recommendation to the batter:

1. The situation (i.e. the count)
2. Some way to predict the result of a hit given a pitch type
3. Data about the pitcher
4. Data about the batter

We note that we will never recommend a player to sit on a pitch if there are currently two strikes; this would incredibly unwise as sitting on a pitch implies to not swing even if you know it will be a strike. This would result in the batter being out.

[1] See https://www.mlb.com/astros

2.1 Run Expected Value of Putting the Ball in Play

As stated above, we need a framework with which we can evaluate how productive a player can expect to be when hitting a certain pitch type. Not only are certain batters overall more productive than others, but also certain batters are more productive with certain types of pitches than others. Thus, we utilize a metric called "Run Expected Value". The metric maps a given player and pitch type to the expected number of runs that putting that ball in play will generate. As a pure hypothetical, the Run EV of Cody Bellinger hitting a four-seam fastball could be 0.231. This would mean that, on average, the Dodgers can expect 0.231 runs every time Cody Bellinger puts a four-seam fastball in play.

While these Run EVs are proprietary and developed by 4APP, an MLB analytics consulting firm, they are developed through the following methodology:

- For a given pitch type, a batter's spray angle, launch angle, and exit velocity are measured.
- Given this information, the algorithm predicts the probability that putting the ball in play will result in a single, double, triple, home run, or out.
- Using the run value of these outcomes, a linear combination can be created to determine a batters Run EV for a certain pitch type.

We also note that since we are not recommending a batter to sit on any count where there are two strikes, the Run EV of a 2 strike count is calculated independently of pitch type. Given a pitcher and the current count with 2 strikes, we calculate the Run EV by: $Pr(\text{SO}) \cdot RV(\text{SO}) + Pr(\text{Hit in play}) \cdot RV(\text{Hit in play}) + Pr(\text{Walk}) \cdot RV(\text{Walk})$ where "Pr" is probability and "RV" is run value.

These values allow us to judge the productivity that a player can expect to create by hitting a certain pitch type.

2.2 The Pitcher

In order to accurately predict the tendencies and precision of a pitcher, we must utilize the following data:

```
For every pitcher in the MLB:
    For every count:
        For every pitch type:
            Store the "Usage Percentage"
            Store the "Strike Percentage"
```

In total we aggregated the data from 980,796 pitches in the 2020 MLB season to generate these values.

2.3 The Batter

In order to accurately predict the tendencies and productivity of a batter, we must utilize the following data:

```
For every batter in the MLB:
    For every pitch type:
            Store the "Number of Observations"
            Store the "Number of Balls in Play"
            Store the "Run EV" of putting the ball in play
```

We used the same 980,796 pitches (albeit different metrics) in the 2020 MLB season to generate these values.

3 The Function

Now that we have all the necessary data, we need to think about the algorithmic design of our function. On the surface level, we think about the function as a black box:

- **Input:** A pitcher, a batter, and the current count
- **Output:** A pitch type that the batter should sit on

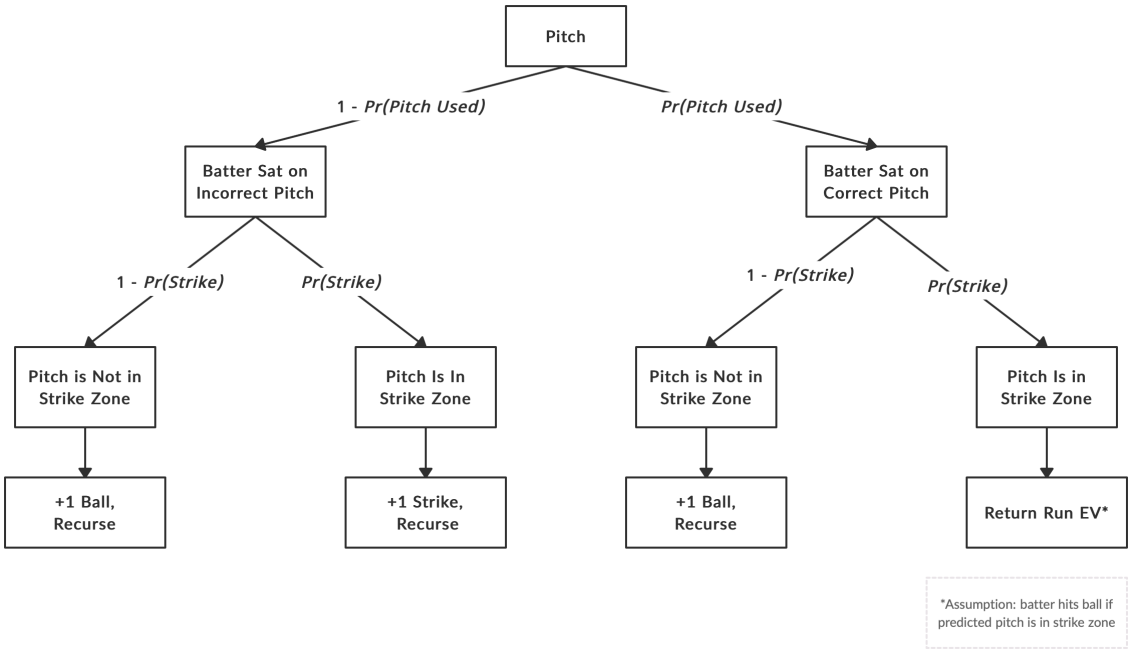
Next, we move onto actually evaluating a how the algorithm should make this decision. As mentioned earlier, we utilized the Run EV metric by 4APP to measure expected productivity. We make the objective of the algorithm to **maximize** the batters Run EV.

Next, we think about how sitting on a pitch affects your productivity. We must account for the fact that if we tell a batter to sit on a certain pitch type, they will perform better if they end up receiving that pitch type. In our model, we accounted for this "boost" in productivity by including the following assumption: if a pitcher is sitting on a certain pitch type and they receive that pitch type, they will make contact with the ball if it is in the strike zone. If the pitch is outside the strike zone, the player will not swing. We are not saying that they will hit the ball particularly well, but rather that they will make contact (the Run EV metric will account for variation in hit quality). We find this to be a fair assumption given that sitting on a pitch allows a hitter to do a much better job of predicting where the ball is going to end up.

We next consider the three possible conditions in which we arrive at a Run EV:

1. The pitch type that our batter is sitting on in is pitched and is in the strike zone. The batter hits the ball (see assumption above). We return the Run EV for that pitcher and pitch type.
2. There are 2 strike on the count. See Section 2.1 for a description of how the EV is handled in this case.
3. Walk. We return a Run EV of 0.25 (the standard walk run value).

If a pitch does not result in one of these three outcomes, we update the count and recurse on the function again. This results in the following decision tree:



We refer to this function as CALC_EV, which takes as inputs the pitcher, batter, count, and pitch to be sat on and outputs the Run EV. Using this, we can now select the pitch type which will maximize the hitters Run EV. We perform the following algorithm to determine which pitch the batter should sit on:

```
def predict_pitch(pitcher, batter, count):
    for pitch_type in pitches:
        EV = CALC_EV(pitcher, batter, count, pitch_type)
        store EV in all_EVs
    return max(all_EVs)
```

Sanity checks confirmed that our function returns logical and intelligent pitch types for batters to sit on. We were incredibly happy with this result and look forward to sharing our framework and algorithm with baseball analytics firms in the near future.

We note that the function inherently cannot be back tested since we cannot speculate on what would have happened had a player been sitting on a certain pitch type.

4 Potential Improvements and Conclusion

First, the run time of the function could be improved significantly if memoization was utilized to store previously calculated values. Currently, the function recalculates many data points on every recursive call. We ended up not implementing this as the run time was actually very fast, but it may become necessary should the model become more complex. Next, we brainstorm the following three potential structural improvements:

- **Looking at more metrics.** The model could be improved to consider metrics such as spin rate, miss percentage, chase rate, fair ball rate, fly-ball rate, line-drive rate, pop-up rate, etc.
- **Using machine learning techniques.** Further stochasticity could be added to model, clustering could be utilized to identify specific pitcher/batter combinations that would greatly benefit from the algorithm, or ensemble methods could be used to combine this model with other common baseball models.
- **Considering broader situational information**. Looking at the current players on base or overall game situation may affect the outcome of the function. For example, the Late-inning Pressure Situation (LIPS) metric could be used to help account for this.

The possibilities for implementation of the algorithm are endless. One can image a world in which a batter wears a smart bracelet on his arm that instructs him which pitch type to sit on before every pitch. One step removed from our analysis could be to potentially reverse engineer the algorithm to instruct the *pitcher* which pitch type *not* to throw. Nonetheless, the utility of this function is both inherent and impressive.