Sabermetrics: Aguirre's Wager Point System

By: Brandon Aguirre (Spring 2019)

https://brancisco.github.io/sabermetrics-2k19-project/#/about

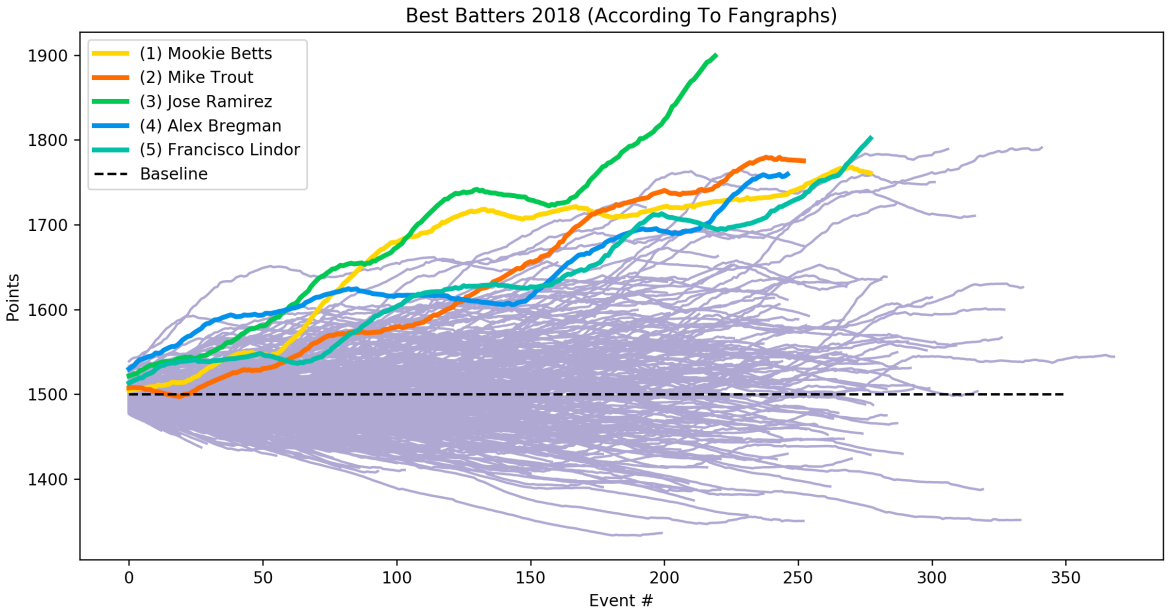

Figure 1. AWPS rating highlighting the top 5 best batters (according to fangraphs.com) plotted against all other players in the MLB.

What is AWPS?

AWPS is a closed point rating system that I have created and implemented for my Sabermetrics class. I define a “closed point” rating system to be a rating system that distributes a finite number of points to all players in the MLB. These points will be transferred between pitchers and batters each time a pitcher and batter face off.

I was inspired to create AWPS by the Hungarian-American physicist Arpad Elo who invented the Elo Rating System. His system was used to rate chess players based on their relative skill levels. The Elo Rating System was a bit simpler to implement because the points were transferred between two players who played the same position (both are chess players). The point system I have implemented transfers points between two players who play different positions - pitcher and batter - and will have to take into account different events.

What does AWPS measure?

The goal of AWPS is to measure how well a player is performing, at batting or pitching, compared to all other players. Similar metrics might be something like Batting Average or Slugging % for batters, and Strikeout % for pitchers. While those metrics are good for measuring the overall performance of a player throughout their career, AWPS is more sensitive to recent activity from a player.

For example say that a player has a small injury, or something going on in their personal lives that is affecting their game. They might have a local minimum in their performance compared to the rest of their career. AWPS is great for this, because the more events a player loses, the larger the amount of points they lose.

What constitutes a loss?

- For a batter, a loss is getting struck out.
- For a pitcher a loss is allowing a hit, or home run.

An example of this sensitivity is Brandon Belt. While he was performing well at the start of 2018, unfortunately he underwent an appendectomy in June, 2018. He also suffered a knee injury later in the season. Examining Brandon Belts AWPS score, it seems obvious he had some issues throughout his season.

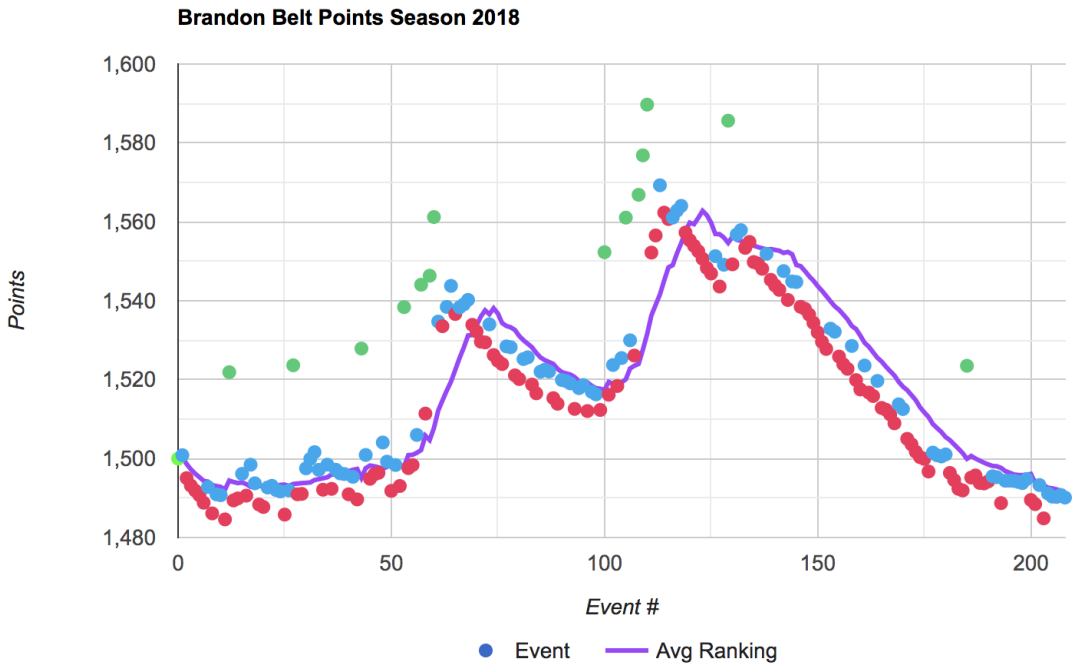

Figure 2. Brandon Belt's AWPS score for 2018.

What are the benefits of using AWPS?

There are two major benefits to using AWPS over other statistics like BA, SLG%, K%, etc..

1. The sensitivity I mentioned in the previous section.
2. AWPS weights each face-off between two players based on their current points.

Since I mentioned how the sensitivity of AWPS is great for watching trends for players, I'll just cover the second point here. AWPS doesn't see things so black and white. For example a hit isn't the same as any other hit… and a home run isn't just any old home run. If a lower ranked batter gets a hit on a high ranked pitcher, they will gain more points than playing someone their own rating. The same goes for the opposite case. A player who has a lot of points isn't going to gain much from getting hits against a low ranked player. This allows for AWPS to adjust a players points more accurately than trying to quantify how good a player is based on the information from just how many hits or home runs they got. The same goes for pitchers.

How do you calculate AWPS?

AWPS is a statistic that needs to be calculated sequentially. Each exchange in points between two players is dependent upon all of the interactions each of those players had before.

The Algorithm

The general algorithm for calculating AWPS is as follows

1. Calculate probabilities of events from a previous season (theoretically these probabilities are changing constantly)
2. Initialize all batters and pitchers with a array containing an initial value
3. For each event between a batter and a pitcher
    - Get the batter and pitchers most recent score
    - Calculate how much each player should wager based relatively on their latest scores
    - Multiply their wager by a factor K (K depends on their scores, independant of the other players score)
    - Multiply their weighted wager by the probability of the current event (hit, home run, strikeout)
    - Depending on who wins and loses, add and subtract points respectively
    - Append this new score to each players array of scores

The wager function

The wager function is essentially a logistic curve. This function was used in the original Elo calculations. It was chosen such that the difference in scores for two different players would decide their respective expected score outcomes.

```
def wager(p, b, k):
  """
  Calculates the points a pitcher, and a batter should wager on a giv
  @param p float: score of pitcher
```

```
    @param b float: score of batter
    @param k int: wager factor, weights how much players should wager
    @return touple (float, float): (pitchers wager, batters wager)
    """
    ep = (10**(p/400))/((10**(p/400)) + (10**(b/400)))
    eb = 1-ep
    return (round(k(p)*ep, 3), round(k(b)*eb, 3))
```

The K factor

The K factor is suggested to be many different values by different organizations in the chess community. They usually base it on some logistic scale, and then stagger the K factor into (usually three) different set points.

Rather than using differnt set points to change the K factor, I decided to come up with a function that would adjust the K factor for every possible point. Below is the function I used to calculate the K factor.

```
def k_function(p):
  """
  @param p float which is the number of points a player has
  @ return float which is the k multiplier for how much a player shou
  """
  res = np.exp(3600/(p))
  if res >= 20:
    return 20*2
  elif res <= 10:
    return 10*2
  else:
    return res*2
```

I designed this function to resemble an exponential curve that lined up close with suggested values from FIDE.

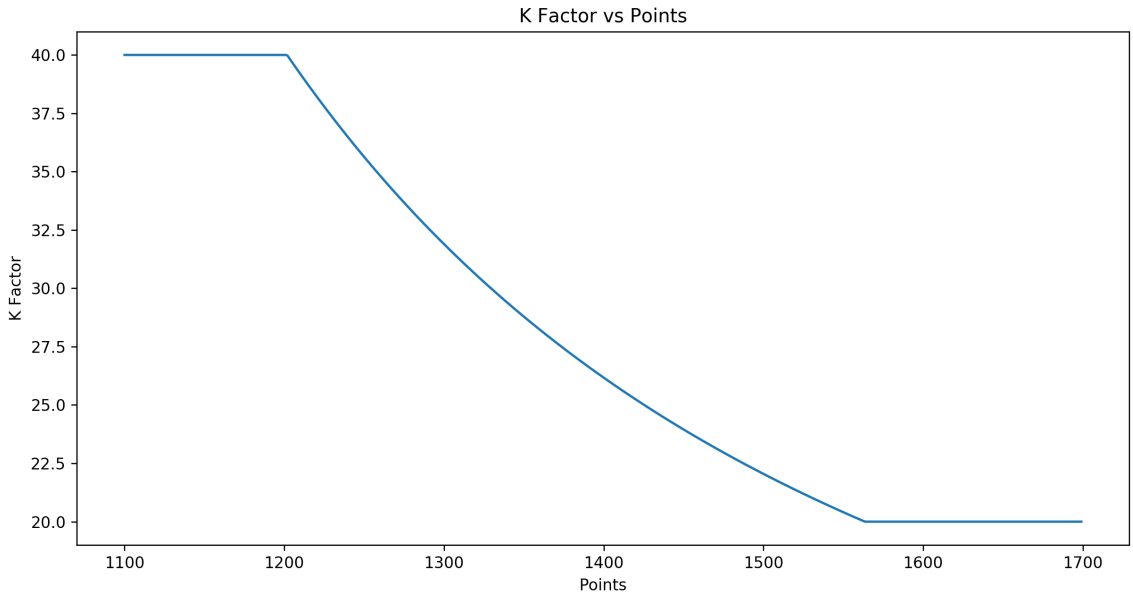

Figure 3. K factor function calculated with the above code.

The event probabilities

Calculating the probabilities for an event is straight forward for my initial AWPS scoring. However, the probability of these events changes every time batters and pitchers face off. If AWPS was used in practice, it would be optimal to use some type of weighted average function to calculate average probabilities over some time period.

```
def get_proba_function(df):
  """
  Create a function which produces the probability of: a hit (excludi
  @param df pandas.DataFrame which has a years seasons worth of statc
  @return a function which takes in an event (h, hr, k), and returns
  """
  n_1b = len(df.loc[df.events == 'single'])
  n_2b = len(df.loc[df.events == 'double'])
  n_3b = len(df.loc[df.events == 'triple'])
  n_hr = len(df.loc[df.events == 'home_run'])
  n_so = len(df.loc[df.events == 'strikeout'])
  emap = { 'h': n_1b + n_2b + n_3b, 'hr': n_hr, 'k': n_so }
  return lambda e: emap[e] / sum(list(emap.values()))
```

Okay, so does AWPS really work?

With my current algorithm and calculations, AWPS does seem to be working! However, there is definitely a lot of room for improvement, and many ways this could be expanded. Below I'll show some graphs that indicate AWPS is doing what it is supposed to.

Distributing the points to players throught a season

After running the algorithm on all of the Statcast data from 2017 and 2018, I wanted to see how the points were distributed between pitchers and batters.

Distributing points evenly between pitchers and batters. was challenging for two main reasons:

1. There are many more batters than there are pitchers.
2. Pitchers and batters are doing two fundamentally different things, so dividing the points evenly requires more than just a win or a loss (we need to answer the question "How much did they win by?").

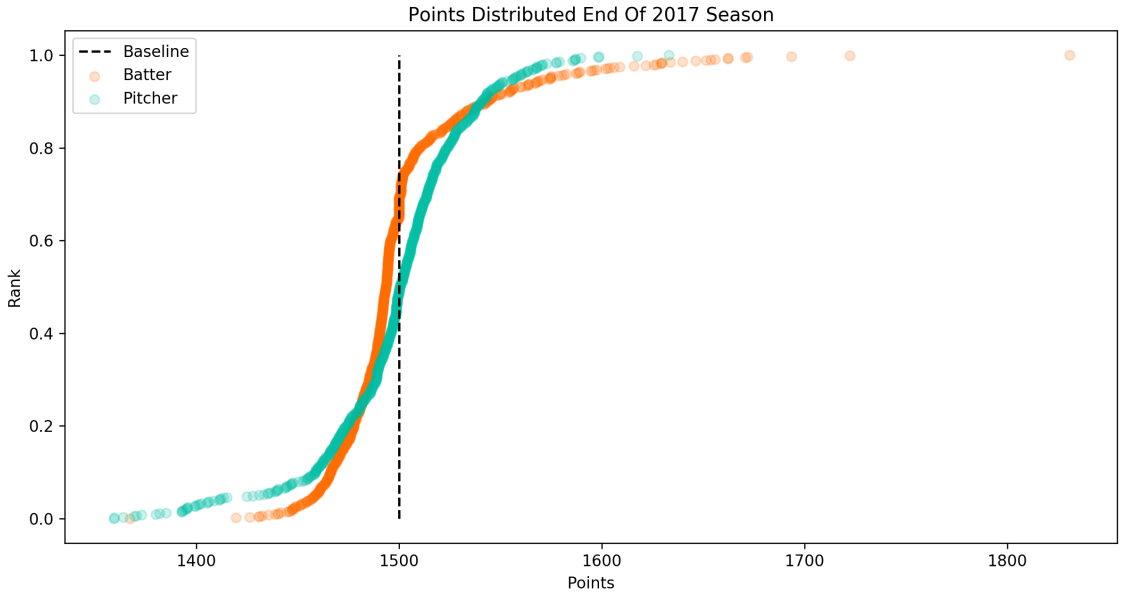

Figure 4. Some observations above: There are relatively more pitchers who cross the 1500 points baseline. There are more batters who have a higher number of points. There are more pitchers with a the lowest number of points.

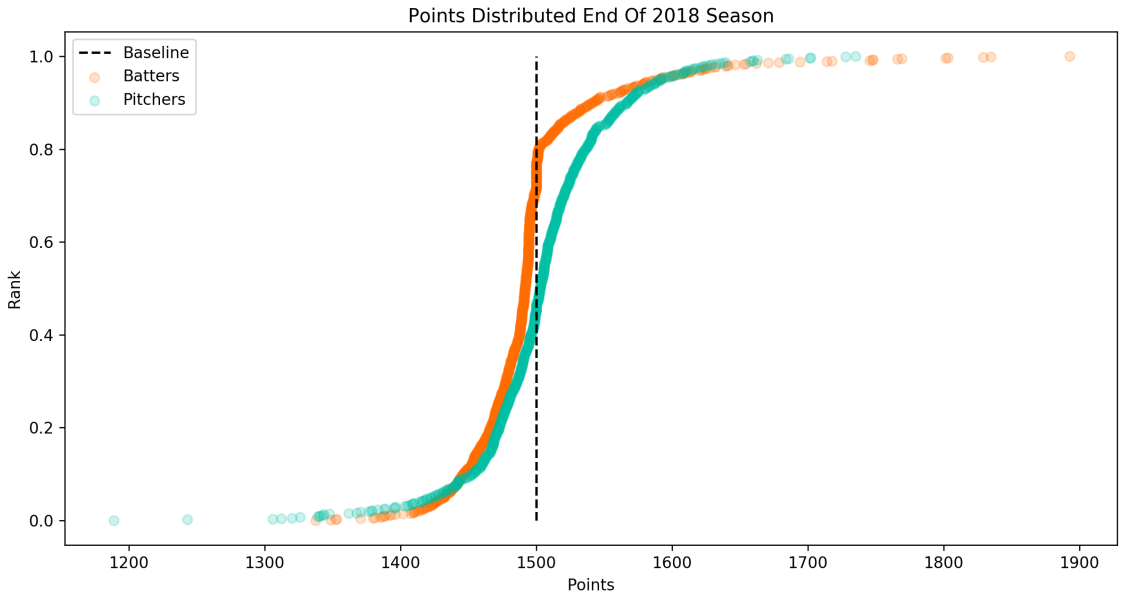

Figure 5. The 2018 data has a very similar distribution as the 2017 data which is expected. It is interesting though that the tails seem to line up more for the 2018 data.

With some more effort, I think there should be a way to assign points that creates a distribution where the pitchers and batters line up more. Although it could just be the nature of batting vs pitching.

Comparing AWPS ratings to fangraph ratings

In Figure 1. I showed the AWPS rating for the top 5 batters according to fangraphs 2018 leaderboard. The players are clearly amongst the very top rated out of all of the players in the AWPS system. Below is a graph of the AWPS scores of the top 5 rated pitchers by fangraphs.

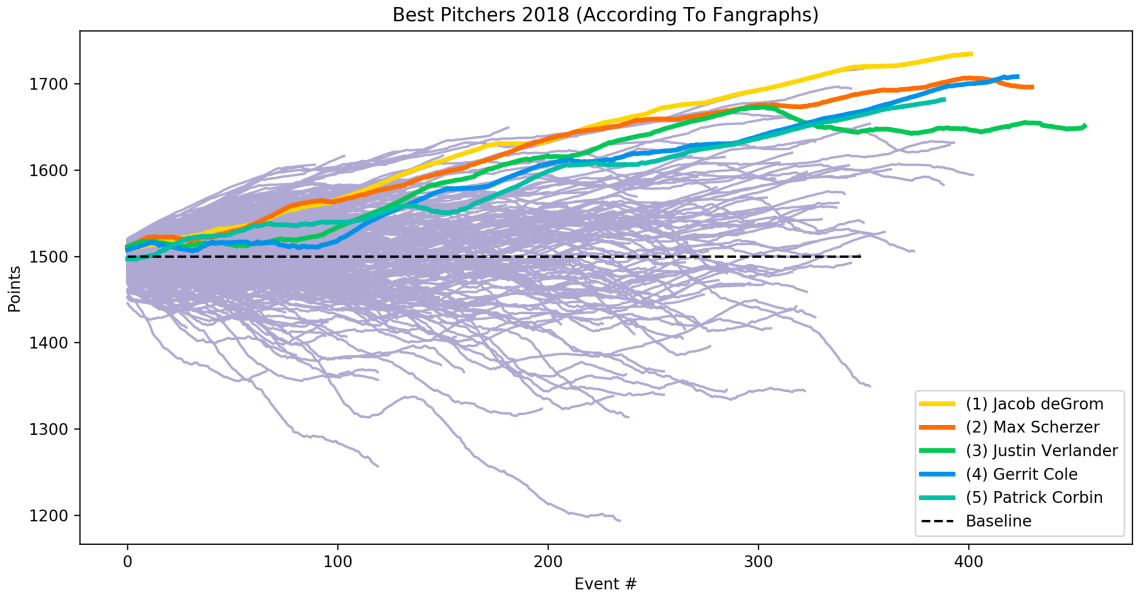

Figure 6. AWPS rating highlighting the top 5 best batters (according to fangraphs.com) plotted against all other players in the MLB.

I am pretty happy with these results, as the top ranked pitchers and batters are pretty consistent with fangraphs ratings!

Issues with AWPS

Stale players

But what happens when players get injured and stop playing?! What happens when players retire!? What happens when players are mostly on the bench?!

Further work in developing the AWPS algorithm is definitely needed to make it a more viable stat no doubt. One solution could be to monitor inactivity between playing. If a player is taking a break or their face-off frequency drops, some of their points could be deducted and thrown into a "pool" of sorts. Those points could be distributed to other players who are staying more active.

Many other events besides just hits, home runs, and strikeouts

Some pitchers have many more BB than others. Some batters get on base from their hits more often. Some batters are clutch in situatins like when the bases are loaded.

Adding in more events makes the probability calculations a lot more complex. However, taking into account more of these stats could make AWPS much more robust.