DraftKings Fantasy Basketball Optimizer

Optimizing Daily Fantasy Basketball Team Based on Momentum

Shane Fenske – CPSC 458 – 12/20/2015

Introduction

Last Spring I became interested in daily fantasy basketball. I used the site DraftKings.com to enter daily contests where you assemble a team of eight NBA players and score according to how those eight players perform in their actual NBA game that day[i]. The contests often times cost money to play and offer prize money to high scoring competitors. Despite only playing for a few dollars a day, I found myself spending an hour each day attempting to come up with a lineup I thought could win each night. Eventually, I grew tired of spending so much time on something for which I had so little at stake. However, I knew a computer could do a much better job algorithmically implementing the strategy I was using by hand.

Strategy

My strategy was to target players that had performed very well relative to their fantasy salary in their past two games. In this program, I couple that strategy with a version of the momentum stock trading strategy we implemented in class. I form a projection for a player based on the average fantasy points they scored in their previous two games. The player is only eligible for my lineup if he has positive scoring momentum. This means he is only eligible if he has scored more in his most recent two games than in the two previous. With this projection, the optimizer finds the highest projected lineup that meets the criteria of having 1 point guard, 1 shooting guard, 1 small forward, 1 power forward, 1 center, 1 guard (point guard or shooting guard), 1 forward (small forward or power forward), and 1 utility player (any position). Furthermore, it must meet the criteria of the 8 players combined salaries being less than the $50,000 salary cap.

Implementation

The optimizer consists of 3 programs: scrape.py, optimize.py**,** generate.py. The user only calls generate.py. First I will explain the function of the first two programs and then I will explain the ways in which generate.py can be called.

scrape.py: Not to be called by user. Walks from DraftKing's NBA lobby into a contest and downloads that day's lineup of available players and their salaries into player_data.csv. It then searches for every player in this csv on basketball-reference.com, enters their individual page, and writes to player_data.csv the data it finds on their previous 4 games. Each player's fantasy projection is written as the average of their last two games in the column 'last two.' If they scored more in the two games prior to their most recent two then they are removed from player_data.csv as we do not want players with negative momentum.

optimize.py: Not to be called by user. Uses scrape.py generated csv to print best lineup. First creates list of all players playing that night, the list is sorted by a value column which is equal to the player's projection / (their salary / 1000). Then determines best lineup by populating the guard, forward, and utility positions with the first 3 players in the sorted playerlist who can legally fill the 3 positions. To fill the 5 remaining positions (PG,SG,SF,PF,C), the program brute force tries every legal combination. It then prints the best lineup, unless that lineup is projected to score less than 270 points (270 is the safety threshold point I determined where larger scores could usually win DraftKings 50-50 contests). Optimize.py can alternatively just print the best value players at a specified position as explained below.

generate.py: Main function to be called by user. Calls the other two programs dependent on what arguments it is passed. Called in this way:

```
% python generate.py
```
calls scrape.py (which can take up to 10 minutes) and then optimize.py. Prints optimal lineup.

```
% python generate.py  player_data.csv
```
runs optimize.py on csv specified (csv must have been previously generated by scrape.py). Prints optimal lineup..

```
% python generate.py  player_data.csv position
```
Prints list of players in the specified csv sorted by value[ii] . List include only players who play specified position.
position options are: "powerforward", "smallforward","guard", "pointguard", "shootingguard, "util", "forward", "center":

Conclusion

In the one night of real contest tests I did (12-21-2015), my optimizer produced the following lineup:

```
             Chris Paul  PG Projection:   47.9     $8100 Value 5.99
           Jimmy Butler  SG Projection:   43.5     $7400 Value 6.21
          Tobias Harris  SF Projection:   35.5     $6500 Value 5.92
       LaMarcus Aldridge  PF Projection:  37.65     $6800 Value 6.27
              Pau Gasol   C Projection:  51.15     $7600 Value 7.31
            Jeremy Lin  PG Projection:  41.65     $5200 Value 8.33
          Arron Afflalo  SG Projection:  32.65     $4900 Value 8.16
           Kyle O'Quinn  PF Projection:   22.9     $3400 Value 7.63
```

This lineup reveals the fundamental flaw with using an algorithm to pick a fantasy lineup. Kyle O'Quinn injured his ankle the game before and was not going to play on this night. Furthermore, my algorithm had no idea that recent injuries to Portland Trailblazer guards Damian Lilliard and CJ McCollum meant backup guard Tim Frazier was going to play a lot more than he had in the previous two games. With O'Quinn and Frazier's situations in mind I was able to add Frazier and drop O'Quinn and keep the majority of the algorithm-generated lineup intact. Providing a forward replacement for O'Quinn is where calls like "python generate.py player_data.csv forward" can come in handy. My slightly modified lineup scored over 270 points and placed in the top half of the contest I entered.

Although I will still need to double check the output of this optimizer for injury issues, I deem this project a success as it seems this system will allow me to participate in daily DraftKings contests using a reasonably competitive lineup without wasting an exorbitant amount of time deciding on my lineup. However, the system could be improved. Some of the improvements I hope to make going forward include:

- Implement a linear model to predict a player's points based on the players recent performance, the projected pace of play for the game, and their opponent's defensive results against that player's position.
- Daily email report with lineup information for that night's games with a recommendation of whether to play or not.
- Avoid large daily scrapes by database-ing a large scrape and then adding to it using smaller scrapes of each day's box scores`
- Generate two separate lineups each day – one lineup up of players with a high projected worst possible outcome for the day's game and one with the highest projected potential. The first would be used in low risk low reward "50-50" contests where all competitors finishing in the top half of the competition win the same prize equal to about 190% of their buy-in. The latter would be used in the site's high risk-high reward type contests where an exceptional performance is needed to earn money, but the money earned is often multiple times larger than the buy-in.
- Add option to help you remove players you don't like or know are hurt from optimization

---

[i] Scoring System:

Point = +1 PT Made 3pt. shot = +0.5 PTs Rebound = +1.25 PTs Assist = +1.5 PTs Steal = +2 PTs Block = +2 PTs Turnover = -0.5 PTs
Double-Double = +1.5PTs (MAX 1 PER PLAYER: Points, Rebounds, Assists, Blocks, Steals)
Triple-Double = +3PTs (MAX 1 PER PLAYER: Points, Rebounds, Assists, Blocks, Steals)

[ii] the player's prejection / (their salary / 1000)