Daily Fantasy Sports Forecasting

Steven Miller

Abstract

The goal of this paper is to assess the feasibility of using predictive models in conjunction with daily fantasy sports contests. In these contests, points are awarded to players based on their performance during sporting events. Contests typically take one of two forms. First is a guaranteed prize pool (GPP) where a smaller number of entrants can win prizes, which scale based on the performance of the entrant (the highest scoring team wins the largest prize with prizes decreasing in value for lower-scoring teams. The second form is a head-to-head, or fifty-fifty contest. In these contests, half of the entrants will win, and the prize is the same for all winning participants. The winner and the person who ranks in *p/2* where *p* represents the total number of participants will receive the same prize.

By using available historical data, the aim of the project is to build a predictive model that can accurately predict the number of points scored by each player during an event. Using this data, optimized fantasy lineups can be generated and entered into contests. This will be considered a successful project if the results generated a positive expected outcome across the test data.

The Data

For this project, data has been acquired from several sources. The first source is Basketball Reference. This website contains a database of nearly anything anyone could want to know about the sport of basketball. Using the Python library *basketball_reference_web_scraper*, I was able to collect individual player statistics for every game for the entire 2017-2018 NBA season.

The second dataset utilized for this project was retrieved from Kaggle. This was a user-aggregated dataset that contained the results of about a month's worth of contests on the Daily Fantasy Sports website DraftKings. This data contains information on each player's performance, the contest standings, and the payout structure of the contest. This data will be used to measure the effectiveness of the model-generated lineups.

Process

The ultimate goal of this project, if predictive modeling is found to be a feasible tool, is to create a full pipeline where raw sports and contest data is fed in, and optimized lineups for a contest are generated for entry. To make this transition from project to production seamless, effort was made to create code resembling such a pipeline from the start. This also assisted in the ability to test predictions against the test contest data for evaluation.

The first step of the process, as with almost all data science projects, was the gathering of raw data. Using the *basketball_reference_web_scraper* Python package, I was able to gather records of all NBA games from the 2017-2018 season. Thanks to Basketball Reference's scraping-friendly policies, this was an easy task. They only ask that data not be collected any faster than a human would reasonably use the website. In order to comply with this request, a sleep function was included in my scraping script to slow down the process. As the data was downloaded from Basketball Reference, it was then stored in a PostgreSQL database for easier access later. In addition to this data, a collection of contest results was gathered from Kaggle that represented most dates from November 2017. As this data was not going to be used in a way that would require frequent querying, it was left in a flat, CSV state.

The structure of fantasy sports contests assigns each athlete a salary for the event. Entries in the contest will have a salary cap to develop a team of players while remaining within. This leads to different strategic decisions that can be made in the formation of a team. Entrants could choose to meet the salary cap by choosing a team of average players, or they could pick a couple of all-star players and use lower-caliber players to fill out the rest of their team. One goal of the predictive model will be to identify players who have higher value than other players, where their expected points per dollar of salary is above the average for their position.

Statistic	Points
Points	1
Rebounds	1.25
Assists	1.5
Steals	2
Blocks	2
Turnovers	-0.5
3-points made	0.5
Double-Double	1.5
Triple-Double	3

Figure 1.1 - Points earned per statistic for NBA DraftKings contests

In order to develop a benchmark for the predictive model, an initial measurement to outperform, sample lineups were generated using the *pydfs-lineup-optimizer* library using season-to-date fantasy point averages in place of a projected value for each player. These values are the same as the ones presented to users when creating a lineup within the DraftKings website. Using these values, twenty lineups were generated for each contest with available data and actual outcomes were compared with the contest results. In total, 4,700 lineups were entered in 235 contests, resulting in entry fees totaling $57,687.60 and theoretical winnings of $8,104.50, a net loss of $49,458.10. These results are pretty terrible, but not unexpected. One of the challenges of this project is that fantasy sports are a negative-sum game. A number of players each pay an entry fee to participate in a contest. Somewhere between 10 and 20 percent of this fee is kept by the platform hosting the contest, with the rest paid out in winnings to the entrants. As a result, even “average” performance over time will result in losing out in the long-term.

After developing this initial benchmark, as well as a reusable pipeline for further backtesting, I moved on to developing actual predictive models. For inspiration, I took a look at some websites that currently

specialize in predicting fantasy points. I wanted to get an idea of what values they appear to be considering in their projections. The website I chose to look at initially was Daily Fantasy Nerd. I found that they considered a large number of factors in their projections:

- Days rest - how many days the player has had since their last game
- Points Scored - How many points are expected to be scored by the players team according to Vegas lines.
- Usage - what percentage of a player's team that the player is involved in
- Player efficiency rating - a productivity rating of the player
- Opponent pace - On average, how many possessions to opposing team uses in a game
- Opponent defensive efficiency, how many points are typically scored against the opposing team per 100 possessions
- Defense vs. Position - how many fantasy points are scored by players in the player's position against the opposing team compared to the league average
- Field Goal Attempts - how many attempts a player made to score in the past 2 games, past 5 games, and all season.
- Minutes - how many minutes a player played in the past two games, past 5 games, and all season.
- Fantasy Points - how many fantasy points the player has scored on average in the last 5 games and all season, as well as ceiling and floor values based on the standard deviation of the player's fantasy points.

With this knowledge, I set out to build an initial linear regression model. I don't currently have historical vegas data, so some of the Daily Fantasy Nerd data was impossible to recreate. As a starting point I created an initial model using days of rest, average field goal attempts for the last two games, five games, and season, average minutes played for the last two games, five games, and season, and average fantasy points for the last two games, five games, and season. The result was a model with an adjusted R-squared

of 0.863. It's worth noting that these results may not be reproducible, as the model was trained on the entire season of data, not all of which would be available for training at the time predictions would be generated for a night's games.

OLS Regreession Results

Dep. Variable:	dk_points	R-squared (uncentered):	0.863
Model:	OLS	Adj. R-squared (uncentered):	0.863
Method:	Least Squares	F-statistic:	1.189e+04
Date:	Sun, 26 Jan 2020	Prob (F-statistic):	0.00
Time:	21:54:04	Log-Likelihood:	-96543.
No. Observations:	26435	AIC:	1.931e+05
Df Residuals:	26421	BIC:	1.932e+05
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Season Average Minutes	-0.2335	0.029	-8.028	0.000	-0.290	-0.176
Season Average DK Points	0.7321	0.034	21.356	0.000	0.665	0.799
Season Avg FGA	0.0367	0.087	0.420	0.675	-0.135	0.208
Season Avg FGM	-0.4509	0.182	-2.482	0.013	-0.807	-0.095
Std DK Points	0.1438	0.021	6.952	0.000	0.103	0.184
L2 Average Minutes	0.2175	0.025	8.686	0.000	0.168	0.267
L2 Average DK Points	0.0356	0.022	1.610	0.108	-0.008	0.079
L2 Avg FGA	0.0080	0.056	0.143	0.887	-0.102	0.118
L2 Avg FGM	-0.1000	0.095	-1.048	0.294	-0.287	0.087
L5 Average Minutes	0.0416	0.037	1.137	0.255	-0.030	0.113
L5 Average DK Points	0.1536	0.036	4.232	0.000	0.082	0.225
L5 Avg FGA	0.3290	0.090	3.669	0.000	0.153	0.505
L5 Avg FGM	-0.2211	0.161	-1.371	0.170	-0.537	0.095
days_since	-0.0233	0.011	-2.129	0.033	-0.045	-0.002

Omnibus:	1093.980	Durbin-Watson:	2.003
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1521.502
Skew:	0.414	Prob(JB):	0.00
Kurtosis:	3.834	Cond. No.	248.

Upon backtesting of this model, I found improved performance in comparison to the initial benchmark, but still not quite to a level I would deem satisfactory. Instead of losing $49,458.10 in the course of a month, the model *only* lost $39,534.85. My next step was to add additional team information, regarding the number of points scored on average by the player's team as well as the team they would be competing against. This performed butter, but still with a loss of $17,092. From here, I attempted to move to an XGBoost model using the same data. The result was surprisingly phenomenal performance, winning nearly every contest and earning over $1 million. Of course, if it sounds too good to be true, it is. My earlier concern about fitting across an entire season of data leading to overfitting wasn't an issue with a

simple linear regression, but XGBoost captured the nuances of the data so well that it was essentially predicting everything with perfect accuracy. It was clear that changes were going to need to be made to ensure a realistic backtesting process. I rewrote by backtesting code at this point to essentially retrain the model for every day being backtested. This made the time needed to execute increase significantly, but resulted in more realistic results. Suddenly instead of winning over a million dollars, I was back to losing, $26,782 to be exact.

At this point I decided that my current approach to the problem was not sufficient enough for success. The results needed weren't sufficient, and I felt the data currently in use was being used to its maximum ability. I looked into two Daily Fantasy Sports websites that offered paid services, Daily Fantasy Nerd, and Fantasy Cruncher. In addition to player data, both sites also offer their own daily projections for each player. Both sites offer the capability to download their data to a CSV file, but also prohibit the use of web scrapers and bots. To comply with their terms of use, I manually visited each page for the 2019-2020 season to date and downloaded the player data. I then stored this data into two tables on my existing database of player data.

Using this newly retrieved data, I sought to build a new model. I started by creating an OLS regression using the two tables of data from Fantasy Cruncher and Daily Fantasy Nerd. When the relevant columns of data were used to predict fantasy points, an adjusted R^2 value of 0.88 was found, which was promising. From there, I moved on to developing a new XGBoost model, followed by a neural network using TensorFlow.

I was now unable to backtest my model on the same data as I had previously used, as all contest data retrieved from Kaggle was focused on the 2017-2018 NBA season and neither website had data going back that far. Instead I had to generate projections for each of my two models for each day, and use a feature on Fantasy Cruncher's website to manually backtest each result. From January 22nd, 2020 to January 31st, I picked larger contests to compare my model's results against. Overall, the TensorFlow

model was more successful in terms of profitability, by managing to win the grand prize in two contests. The XGBoost model was more consistently successful, with a win rate of 52% compared to 41% for TensorFlow. Because of the luck factor involved in winning first place in two contests, I'm not prepared to objectively call the TensorFlow model better and would personally still be more comfortable using the XGBoost implementation.

XGBOOST	Total Winnings	Avg. Win Rate	Total Profit
1/22	406.5	91%	$241.50
1/23	1926.5	37%	$261.50
1/24	323	27%	-$707.00
1/25	7238.5	82%	$2,573.50
1/26	6283	73%	$1,538.00
1/27	4260.5	63%	-$409.50
1/28	7477	60%	$4,197.00
1/29	5990.25	28%	-$319.75
1/30	2453.75	38%	-$2,900.25
1/31	3106.25	33%	-$1,508.75
Grand Total	**39465.25**	**52%**	**$2,966.25**

TensorFlow	Total Winnings	Avg. Win Rate	Total Profit
1/22	474.25	84%	$309.25
1/23	252.25	40%	-$1,412.75
1/24	81	26%	-$949.00
1/25	36802	69%	$32,137.00
1/26	66599.25	67%	$61,854.25
1/27	1524.25	18%	-$3,145.75
1/28	3832	39%	$552.00
1/29	14762	28%	$8,452.00
1/30	4082.75	41%	-$1,271.25
1/31	661.5	14%	-$3,953.50
Grand Total	**129071.25**	**41%**	**$92,572.25**

A breakdown of contest statistics between the XGBoost and TensorFlow models.

As a final attempt to improve on these results, I scraped Basketball Reference one more time, now with data for the 2019-2020 season. I joined this dataset with the Fantasy Cruncher and Daily Fantasy Nerd datasets. Unlike the premium datasets, the Basketball Reference dataset simply contains player logs of each game. For the purposes of modeling, many of the attributes were averages of the past *n* games or season averages. Calculating these for each model made the process much more time consuming, but the hope was that better accuracy would be worth the hassle. If I decide to move forward with this project beyond this point, I'll likely look into calculating these averages when the records are created, rather than calculating them when loading data for modeling.

The final iteration of my model used the three combined datasets to create a set of training data based on the season-to-date results prior to an evening's game. This resulted in a model with {N} features. An OLS regression on this data resulted in an adjusted R^2 value of 0.875. From this point, the training data was fed into a neural network, which produced a mean average error of approximately 5 across the training set. To ensure that overfitting on existing data wasn't tainting the model, projections were generated multiple times, using only data that would have been available prior to an evening's games. From this data, new lineups were generated and tested on the same contests as the previous models.

TF 3	Total Winnings	Avg. Win Rate	Total Profit
1/22	$550.25	83%	$385.25
1/23	$3,980.75	40%	$2,315.75
1/24	$160.25	25%	-$869.75
1/25	$16,590.50	81%	$11,925.50
1/26	$4,719.25	56%	-$25.75
1/27	$2,576.75	28%	-$2,093.25
1/28	$3,280.75	41%	$0.75
Grand Total	**$31,858.50**	**49%**	**$11,638.50**

Results of backtesting on the final model using all three data sources.

Conclusion

Ultimately, I feel like I've gotten as far as is reasonably possible with the projection aspect of this challenge. On average, there was a deviation of about 7.3 points between my projections and the end results. Moving forward, the new challenge is focused more on the strategy behind developing lineups for a contest. Currently, these lineups are using the highest projected combinations, which sometimes results in the same player being on every lineup in a contest. If anything happens to this one player, the entire night is ruined. I feel more can be done strategically to minimize risk and maximize potential profit. I hope to explore this more in the future by testing different parameters for the creation of lineups and comparing the results across a series of contests, as I have done already with projection models.

References

DraftKings. (2019, December 15). Average Results. DraftKings. https://www.draftkings.com/average-results

Earl, J. (2019). Running head: Optimization of Daily Fantasy Basketball Lineups 1 Optimization of Daily Fantasy Basketball Lineups via Machine Learning. https://digitalcommons.liberty.edu/cgi/viewcontent.cgi?article=1909&context=honors

Gehman, R. (2015). How To Make DFS NBA Projections - DraftKings Tutorial (Updated) [YouTube Video]. In YouTube. https://www.youtube.com/watch?v=OxVmp_kZ8Nc

Harwell, D. (2015, October 12). Why you (probably) won't win money playing DraftKings, FanDuel. Daily Herald; Daily Herald. https://www.dailyherald.com/article/20151012/business/151019683/

Haugh, M., & Singal, R. (2018). How to Play Strategically in Fantasy Sports (and Win). http://www.sloansportsconference.com/wp-content/uploads/2018/02/1001.pdf

Hermann, E., & Ntoso, A. (2015). Machine Learning Applications in Fantasy Basketball. http://cs229.stanford.edu/proj2015/104_report.pdf

Ho, R. (2017). Predicting Player Performance for Daily Fantasy Sports | NYC Data Science Academy Blog. Nycdatascience.Com. https://nycdatascience.com/blog/student-works/machine-learning/predicting-player-performance-daily-fantasy-sports/

KengoA. (2019, November 18). KengoA/fantasy-basketball. GitHub. https://github.com/KengoA/fantasy-basketball

Ruths, T. (2015, September 24). Draft Kings and Data Science: Double your Money, Double your Fun - Data Shop Talk. Data Shop Talk. https://datashoptalk.com/double-yo-money/

Schultz, J. (2019, July 28). Optimizing a Daily Fantasy Sports NBA lineup — Knapsack, NumPy, and Giannis. Big-Ish Data; Big-Ish Data. https://bigishdata.com/2019/07/28/optimizing-a-daily-fantasy-sports-nba-lineup-knapsack-numpy-and-giannis/