Fantasy NBA Predictions

Final Report

Olivier DUTFOY : A20364492

August 11, 2016

Contents

1 Introduction

The goal of my project is to efficiently predict which NBA basketball players will have the most fantasy points on a given game or week. While this can be done with almost any sports, I focused on fantasy NBA due to the abundance of websites simulating it and the fact that I have some knowledge of the sport. While this kind of game is most of time compared to gambling, there exist a few numbers of *professional players* who have proven it is possible to win in the long term given the correct algorithms. In the future this project could be adapted to others team sports and with some adaption to individual sports.

The project is coded in **Python** and uses machine learning techniques to provide a prediction.

2 Fantasy Sport Rules

As a fantasy NBA player, your objective is to build a roster composed of real NBA players that will provide you the maximum amount of points. Your roster will then compete against other members of your league and you'll be able to change it at the end of every week through transfers or trades.

There are different websites, each having their own rules in order to compute the fantasy score of a given player. I chose the ESPN fantasy league ([1]) scoring system which is as follows :

- Points = 1
- Blocks = 1
- Steals = 1
- Assists = 1
- Rebounds = 1
- Field Goals Made = 1
- Free Throws Made = 1
- Field Goals Attempted = -1
- Free Throws Attempted = -1
- Turnovers = -1

This is however customizable and the values can be changed quite easily in the code.

These basic values provide a fantasy score which ranges theoretically from $-\inf$ to $\inf$. However in practice the score is barely ever under 0 and almost never exceeds 60. For a given week (which usually includes 2-4 games per player) the range on my data goes from -13 to 165.

3 Data Gathering

I took my data from the NBA website itself ([2]). This data was very convenient because it provides the statistics for every player in every game for multiple seasons.

I considered different methods to fetch this data. I first considered using a web scraper (such as *Beautiful Soup* or *Scrapy*). However I had never used one before and realised it would be much easier to directly use a **cURL** request to gather the data used to fill the tables on the web page.

A first request is made to get the ID's of every player for a given season. This list is then stored in **pickle** file and used for the request of each individual player.

Two requests are necessary for each player. The first one to get information such as the name, age, experience, etc. The other to get the statistics for each game of the season. At least one second wait was added between each request. In the end it takes roughly 15 minutes to gather the data for an entire season (which contains 400-600 players).

Before storing it in a **pickle** file, some useless data fields are removed. Currently some fields are also removed due to lack of value for some players.

11 seasons of data were used from 2005 to 2016 with removal of season 2011-12 because it contained less games and started later.

4 Game Score

4.1 Data Structure

I used as features directly the data fetched for each game which are as follows:

- MIN : (minutes played)
- FGM (field goals made)
- FGA (field goals attempted)
- FG_PCT (field goals %)
- FG3M (3 pointers made)
- FG3A (3 pointers attempted)
- FG3_PCT (3 pointers %)
- FTM (free throws made)
- FTA (free throws attempted)
- FT_PCT (free throws percentage)
- OREB (offensive rebound)
- DREB (defensive rebound)
- REB (rebounds)
- AST (assists)
- STL (Steals)

- BLK (blocks)
- TOV (turnovers)
- PF (personal fouls)
- PTS (points)
- +/- (score difference while on the field)

To which I added several features such as:

- winrate so far
- years of experience
- age
- height
- weight
- number of games played so far
- average fantasy score

For most features my vector would then be composed of the average of that feature over the games played so far. Some features are identical through the entire season for a given player such as the age or experience.

4.2 Approaches

I used both a simple and a Ridge linear regression on both normalized or not features but the results were so similar that I usually stopped making the distinction in the end. I also tried using polynomial features but I could not run it on my computer. The instances have then been used in different ways to generate the training matrix X and the score values vector y. Initially I had long computation times (around 15 minutes) to generate those vectors and therefore stored them in **pickle** files so they'll be usable on other models rapidly. Later on I was able to optimize my code making the computation faster but kept generating the files just in case.

Different approaches were used :

4.2.1 Sliding

The basic and most simple approach. The number of points of a given game for a player are matched to the training vector containing the average of all previous games. This means that if a player has played N games in a season, there will be $N-1$ examples computed for the training data.

4.2.2 Including Location

The next idea was to differentiate whether the game is played at home or away. The features remain the same but their amount is almost doubled. We have the same one as above, plus the averages on the home (respectively away) games played so far.

4.2.3 Including Last Games

This second modification is based on the hypothesis that a player is going to play in a similar way as his last games. The number of features is once again doubled using the averages over a selected number of the player's games.

4.2.4 Including the Position

This is a simple addition taking the position of a player. It consist in adding a small binary vector containing a player's position on the field.

4.2.5 Using Best Players Only

This idea comes from the fact that some players are only going to play a few games or probably perform badly almost the entire season. As a fantasy player, their predictions are therefore uninteresting and removing them would hopefully reduce the error a little. In this case only the selected top performers of the first few month of the season are used to generate the training and testing data.

Note that the approaches above can be combined.

4.3 Evaluation & Results

In order to evaluate the model I use the average error made as well as the maximum error. I first assumed the seasons are independent and used all but one k-folding (testing on one season, training on the rest, repeat for each season) to generate an error value.

My first baseline was using last game's fantasy score as prediction for the next game. However a much more accurate one was the use the average fantasy score so far.

Using linear model with normalized features
B : using binary position, _loc : using location stats, _X using last X games

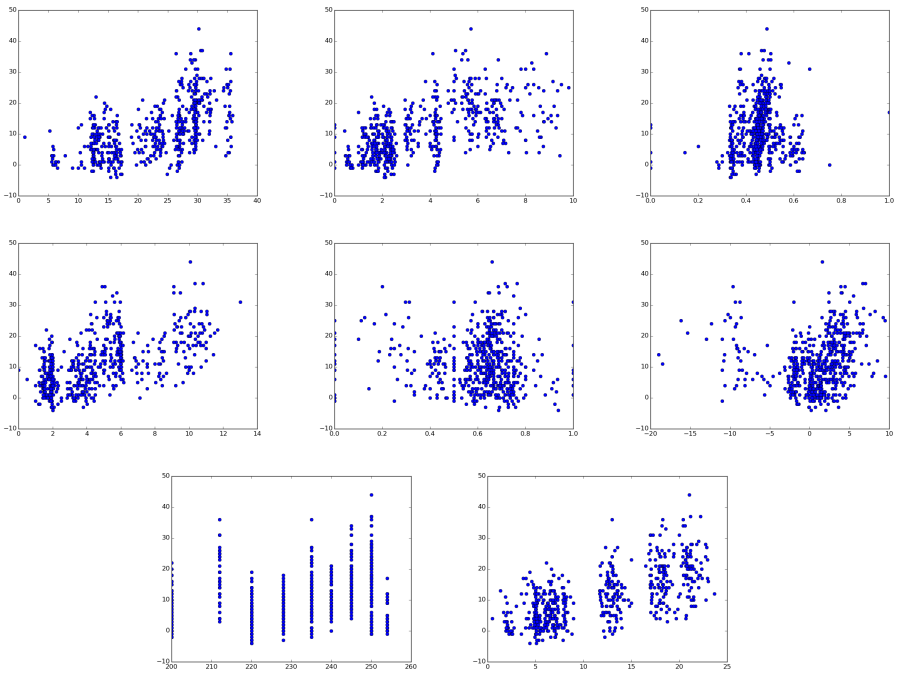
Method		All players	Best 120 players after 1.5 months	
Error	Mean Error	Maximum Error	Mean Error	Maximum Error
Baseline	7.35	48.44	8.88	48.33
Baseline (avg)	5.69	39.62	6.71	38.01
Slide	5.69	38.87	6.66	37.69
Bslide	5.69	38.90	6.66	37.68
slide_loc	5.69	38.81	6.65	37.30
slide_5	5.63	38.47	6.63	37.47
slide_10	5.64	38.13	6.63	37.48
slide_3	5.63	38.56	6.63	37.53
Bslide_5	5.63	38.52	6.63	37.44
Bslide_loc_5	5.64	38.35	6.62	37.04
slide_loc_5	5.64	38.26	6.62	36.94
Bslide_7	5.63	38.30	6.62	36.97

As mentioned earlier, using normalized features or Ridge Regession provides extremely similar results.

4.4 Analysis

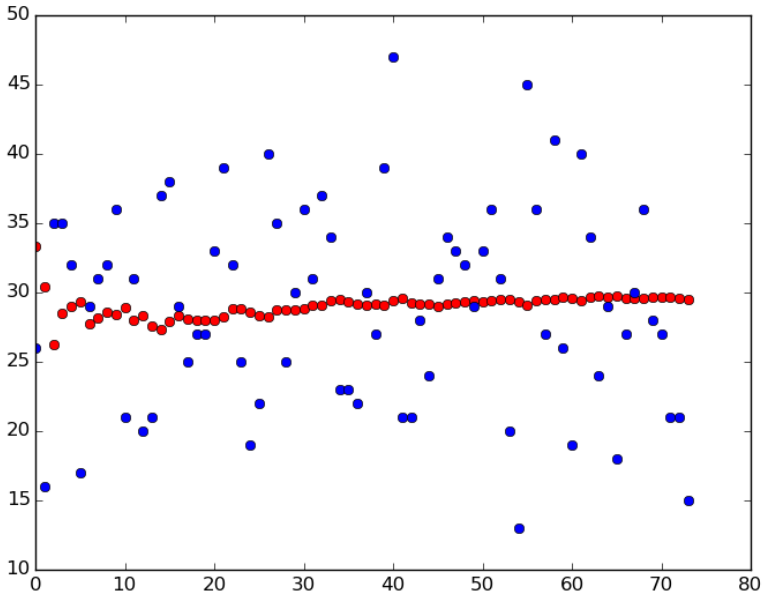
Those results are pretty disappointing considering the improvement compared to the baseline is very small. However there are a few things worth noting. First using the last games seems to provide the best improvement. Second using the best players actually provides a larger error. However the model brings better improvement compared to the baseline. Moreover a deeper analysis would be required, because having a slightly worse mean error is fine as long as the variance is lowered which is likely the case.

It isn't much of a surprise considering the data representation that a linear model would not do the job, here are a few examples on how each feature is correlated to the fantasy score.



y-value : score
from left to right, top to bottom
minutes played, field goals made, FG%, rebounds, winrate, score difference, weight, average score so far

Let's now look at what the predictions look like :



Real values in blue and predictions in red

What is noticeable here is that after a few games the predictions are almost identical which isn't very interesting. Moreover those values are very close to the average score until now which means the model currently brings little to no value.

Considering all this I decided to change my method a little. Instead of predicting points for the next game, I chose to do it on the next given number of days. This has two main advantages :

- It fits the purpose better since fantasy players can (in most websites) draft only once a week. Therefore a prediction for the next week is more interesting than the next game.
- It considerably reduces the amount of training examples (depending on the number of days chosen) which allows the use of polynomial features in reasonable time.

5 Week Score

5.1 Data Structure & Approach

For this method, the same features were used. The only added one was the number of games played the next week.

The approach is very similar except I mostly used the classic sliding due to high computation time if using more features.

5.2 Evaluation, Results and some Analysis

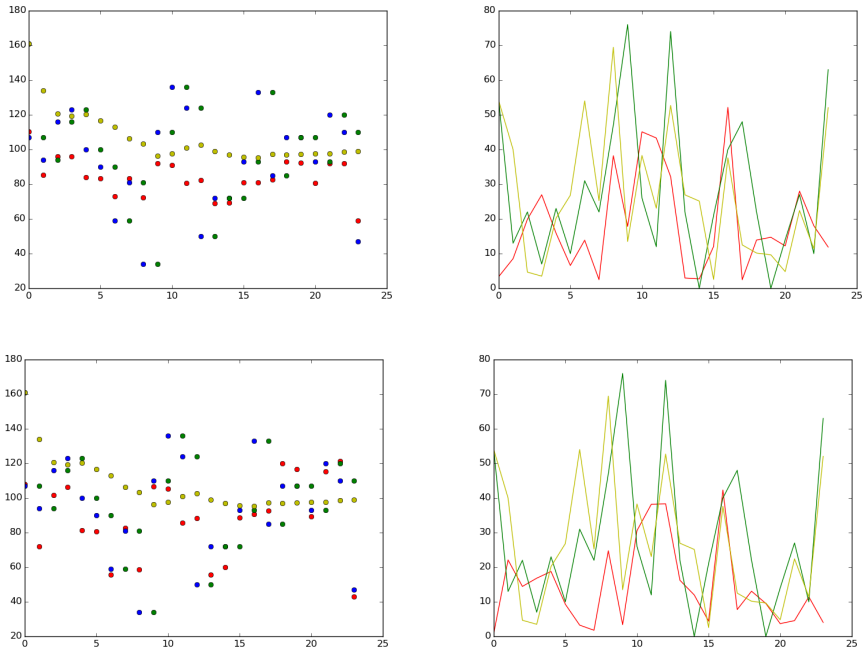
My evaluation is slightly different then previously. This time the seasons are not considered as independent. I implement a simulation system which provides predictions for a given range of days after which the model is retrained and so on. However the initial training set only uses previous seasons. The average error and maximum error is then computed for each week and the average error over all weeks is provided at the end of the simulation.

For the rest of this report every graph shown is color coded:

- blue represents the real value
- red represents the predicted value or the error using model predictions
- green represents the first baseline values (using last week values) or it's error
- yellow represents the second baseline (using average up to now) or it's error

5.2.1 The Use of Polynomial Features

Let's compare the impact of polynomial features. The graphs below show the values for a given player during the 2015-16 season (the range of days used is 7 to simulate a week).



Top : values using linear features and corresponding errors
Bottom : values using polynomial features (degree = 2) and corresponding errors

As expected, the model is better using polynomial features with the errors being smaller even if they peak at the same places. Note that for the entire set of players the average error is 11.24 for linear features compared to 10.02 for polynomial features.

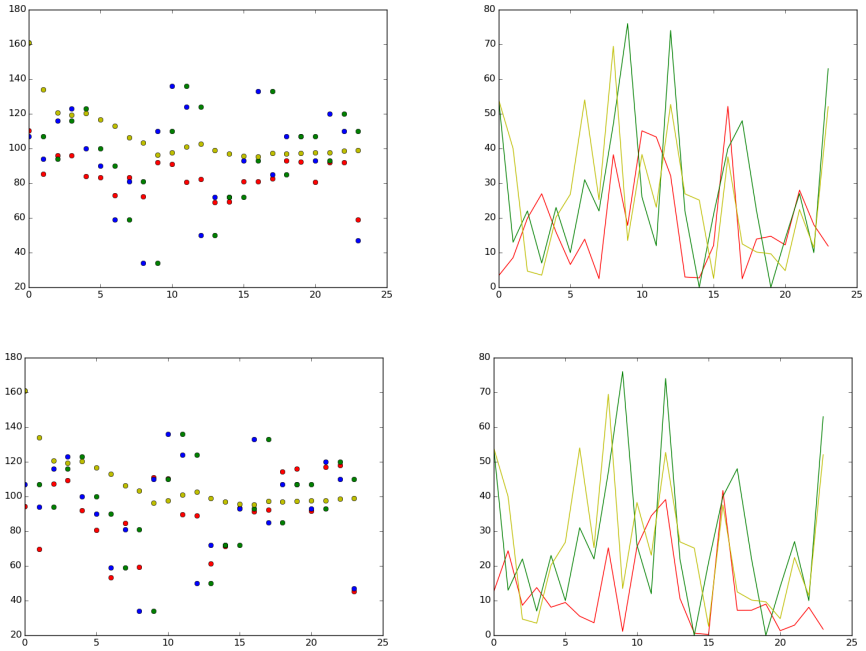
In both cases it is obvious that the predictions vary much more than for the previous method. However what is interesting is the variation for the polynomial features is even higher. This results is some predictions being further off (like on week 2) but most are much closer to the actual value hence the better average.

5.2.2 The Importance of Training Amount

Not worth representing on a graph but it's worth to note that the average error goes gradually from 11.09 when using only one season as training to 10.02 when using all (9 seasons) . It could therefore be interesting to fetch even more data from previous seasons to improve the predictions.

5.2.3 Using Average of Last Games

I wasn't able to run the code with different values due to the very high computing time. The average over the last 5 games were added which using polynomial features takes a lot of time to compute.



Top : values using classic features and corresponding errors
Bottom : values using last 5 games as additional features and corresponding errors

For this player in particular, the error over the weeks is decently improved. However in general, the average error only goes down from 10.02 to 9.95 which is barely worth it considering the computation time. It would be interesting to try and find a way to make the code run faster and try with different

values. It is nevertheless interesting to note that the predictions vary quite consequently when looking at the previous graph which denotes the influence of the new features.

5.2.4 Best Features

Unfortunately I did not have time to fully analyze this. Since I'm using polynomial features, there is over 400 and it is therefore messy to figure out which features are actually important. I have printed the coefficient and there are some interesting values that definitely need some digging into.

5.2.5 Alternative Evaluation

I implemented an alternative way of evaluating my predictions. What a fantasy player really wants is to know who are going to be the best players of next week. I can produce such a list and compare it to the actual best players.

Here is an example output of a 2015-16 week when looking for the 20 best players:

Players Predicted	Predicted Score	Actual Score	Best Players	Score
Blake Griffin	120	144	**Blake Griffin**	144
James Harden	111	134	**James Harden**	134
Paul Millsap	110	133	**Paul Millsap**	133
DeAndreJordan	103	93	**Stephen Curry**	123
Stephen Curry	103	123	**Kawhi Leonard**	118
Andre Drummond	99	92	**Chris Paul**	113
Chris Paul	92	113	**Draymond Green**	104
Al Horford	92	84	**Kevin Durant**	104
Dwight Howard	88	69	Paul George	97
Isaiah Thomas	87	80	**DeAndre Jordan**	93
Greg Monroe	87	68	**Andre Drummond**	92
Jeff Teague	85	84	**Jared Sullinger**	90
Russell Westbrook	85	77	**Anthony Davis**	90
Rajon Rondo	83	70	**Al Horford**	84
Kawhi Leonard	81	118	**Jeff Teague**	84
Draymond Green	80	104	**Isaiah Thomas**	80
Jared Sullinger	80	90	Kevin Love	79
LaMarcus Aldridge	79	58	LeBron James	77
Anthony Davis	78	90	**Russell Westbrook**	77
Kevin Durant	76	104	Reggie Jackson	76
Total	1817	1928		1992

In bold the players who differ from prediction to ideal

While this week was selected because it is the best prediction for the season it still puts forward the potential of the model. Out of 20 players provided, only 4 differ, 3 of whom have close to the worse score. A lot of predictions turned out much smaller than the real values but the model still placed the players in the top 20 which in the end is all that matters. The players wrongly selected still have a decent amount of points which makes the overall results satisfying.

6 Comments

A few things must be taken into account concerning this project.

Some players were deleted due to incomplete data so it would be interesting to try and find a way to use those players.

Some of the data might be slightly incorrect such as player based features. My source only gives a single value but those may have changed over time such as the position of a player.

Given more processing power, some deeper and more interesting analysis could have been done

7 Code Improvements

When I first started this project I was considering a lot what my next steps were going to be to make as an optimal code as possible. However as time went by, a lot of my functions and methods were modified for additional functionalities which sometimes led to some messier code then intended. Here are a few improvements that could be made :

- Rewrite some functions in a simpler way by adding much more smaller functions
- Retrospectively remove patches that were made here and there and recode them in a more coherent way
- Use *dictvectorizer* for my model for a much easier analysis of the important features
- As mentioned before I originally saved my feature vectors in pickle files to avoid having to recompute them again. However this became useless when I optimized my code and could be modified. On the other I don't generate any of those files for my new method but it could prove useful in the future if I complexify my model.
- Use a date library instead of my reimplementation of it
- Restructure my code into more meaningful files
- Delete now obsolete code that I kept along 'just in case'
- Some of my features are redundant and some appear to be useless. Those should be removed and would allow the code to run faster and might even enable the use of more complex models (such as polynomial features of higher degree)
- Write a lot of unit test. I had a lot of errors early that made me lost a lot of time and could have been avoided if I had taken the time to implement unit tests gradually.
- Use *cython* or a way to parallelize the code for faster output

8 Future Improvements

Moving on from here there are a few additions I hope to implement to further improve my results.

I would like to increase the importance of a given player's feature vectors in the training set. As of right now all the players are considered independently and it might prove interesting to change it.

An important addition would be to take team statistics into account as well and especially defensive ones. Currently any game played in the next week is evaluated in the same way and no distinction is made whatsoever.

Another idea would be to use experts opinions. While this was my original objective, it turned out much more complicated than I thought. Finding and exploiting training data is not easy. However looking a next season, I could be using "live" articles and compare different experts articles to my prediction see how it fits in.

I would also like to try using more complicated and maybe better performing Machine Learning techniques.

Finally as I mentioned earlier I would like to further analyze the coefficients of my model to better understand which features are important.

9 Conclusion

After a slow start due to lots of implementation errors I feel like I obtained satisfying results in the end. The first model was pretty disappointing since the results obtained were barely better than a simple average. However with the simulation in place it is possible to obtain a decent list of players.

This project improved my *Python* skills and enhanced my ability to deal with real world data. While I was hoping to gain a little more Machine Learning knowledge, I feel I have a better understanding on how to approach similar problems.

I'm looking forward to improve this model further and try it next season to see how it truly performs.

10 Links

1. http://games.espn.go.com/fba/resources/help/content?name=scoring-settings-custom
2. http://stats.nba.com/