Predicting the Trajectory of an NBA Player's Career

Michael An, Evan Liang, Michelle Zhang

Stanford University

December 16, 2016

I. Introduction

The National Basketball Association (NBA) is the leading men's professional basketball league in the sports industry, with over 2 billion dollars being paid to its players yearly. The vast majority of these NBA players are young, falling between 19 and 25 years of age, with much fewer over the age of 30, and virtually none over the age of 40. For some players, usually superstars, their prime occurs around the ages of 27 and 28. However, the average NBA player spends less than five years in the league, meaning that by their 28th birthday, they'd be out of the league.

Knowing how many years are left in a player's career is a vital piece of information in helping NBA owners decide how they want to make trades and build their team. Should they take the risk and pay for an aging superstar, or will his career end sooner than they expect, leaving the team without money and hope? To help answer this question, we build a machine learning model that given a player and his career statistics, estimates their future productivity as well as when they will ultimately leave the league.

II. Dataset and Features

There are several datasets on NBA player statistics available online, but in order for such a dataset to be useful, we would expect most of the following criteria to hold:

- The dataset contains statistics on players over at least 10-15 years. Since we would like to build a model that predicts future performance of players and longevity given past performance, we need sufficient data over time. Having enough data also allows us to use some form of cross validation to test our model.
- The dataset contains a large number of features (such as game or shooting statistics). The trajectory of a player is likely dependent on many variables, and having too few features can lead our model to underfit.
- The dataset contains a large number of players who have played in the NBA for more than one season. Having past trajectories of individual players, instead of only year-to-year data where the players are constantly changing, is important to separate the effects of an evolving player field and individual career paths.

We found that the player statistics from nbaminer.com satisfied these criteria. The data spanned the last twenty years, with information on nearly 2000 NBA players, and with useful per-season stats such as points per game, rebounds per game, and minutes per game.

The dataset was not available in a particularly convenient form, so we wrote a parser that would extract the relevant statistics for each player. Our final data format was a file for each player, which contains a row for each season that player participated in, and a column for each of the 20 features (corresponding to statistics) in the original dataset. Since different features have different units, they cannot be directly compared, so we normalized all features by linearly compressing each into the interval $[0, 1]$. In other words, we let the minimum value of a feature across any player be 0 and the maximum be 1, and we linearly interpolate between these two extremes.

III. Method

The code to train and test our project was written in Matlab. We have career statistics for $m = 1809$ players. Each player's statistics in a particular season is represented as a feature vector in $\mathbb{R}^d$, where the number of features is $d = 20$. We indexed the players from $i = 1$ to m, and we let p_i denote the ith player. We define $s(p_i)$ to be the total number of seasons played by a particular player p_i, and $\phi(p_i, j)$ as the feature vector corresponding to the jth season played by p_i. For $j > s(p_i)$, we let $\phi(p_i, j) = \mathbf{0}$ since the stats for a player who did not play a particular season can be reasonably approximately by the normalized $\mathbf{0}$. We then let θ be our weight vector, which gives a relative weighting for each feature.

Then, we define the distance between the feature vectors of two players' seasons $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ with a weighted Euclidean norm, as follows:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \theta_i (\mathbf{x}_i - \mathbf{y}_i)^2.$$

This allows us to define the distance between a player p that we are trying to fit, and a player p_x in the dataset that we are trying to compare p to.

This distance between the two players mathematically is as follows:

$$\text{playerDist}(p_x, p) = \sum_{i=1}^{s(p)} \text{dist}(\phi(p_x, i), \phi(p, i)).$$

To predict how a player p would do in the upcoming seasons, we used k-nearest neighbors. Specifically, we found the k players in our training set who had the smallest playerDist to p. Then, for each additional season, if the majority of p's closest neighbors played that season, we predicted that p would play that season as well, and that p's statistics for that season could be estimated by taking the weighted average (based on how close each neighbor was to p) of each neighbor's statistics for that season. If the majority of p's closest neighbors did not play a season, we instead predicted that p did not play that season.

To learn the θ value that minimizes the estimated generalization loss, we used cross validation and coordinate descent on our data set. Doing this involved the following steps:

1. We implement 4-fold cross validation with our training data. We learn on 75% of our training set, and we use the other 25% to test our model's performance. We do this process a total of 4 times, leaving out a different (and disjoint) 25% each time.

2. We take a player p from our test set and randomly select a number of seasons to keep, truncating the rest. Our goal is to predict how many more seasons p played (equal to the number of seasons we truncated), given data on the first few seasons.

3. We then find the k-nearest neighbors of p in the training set, using the playerDist formula defined above. Let $NN_k(p)$ denote the set of k-nearest neighbors of p.

4. Using the majority rule described above, we predict the number of seasons that p played to be

$$\hat{s}(p) = \max \left\{ j \,\middle|\, \sum_{p_i \in NN_k(p)} 1\{s(p_i) \geq j\} \geq \frac{k}{2} \right\}.$$

5. Our estimated generalization loss, which is the average number of seasons that we are off by, can be calculated by

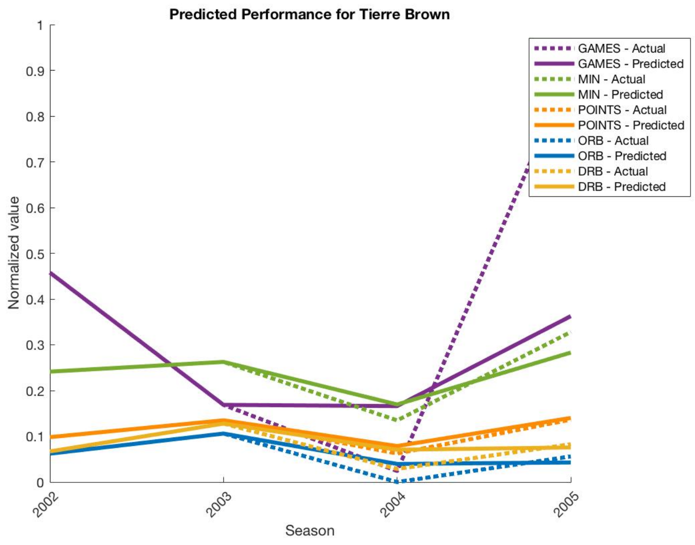
$$\text{Loss} = \frac{1}{|\text{test set}|} \sum_{p \in \text{test set}} |s(p) - \hat{s}(p)|.$$

6. We then perform a coordinate descent update on θ. We first initialize $\theta_i = 1$ for each i. Then, for each iteration, we go through each coordinate θ_i of θ in a random order and update it in the direction (one of $\theta_i - \eta$, θ_i, or $\theta_i + \eta$) that minimizes estimated generalization loss. We used the step size $\eta = \frac{0.5}{\sqrt{\text{iter}}}$ and repeated the process until convergence.

IV. Results and Discussion

Our main focus was minimizing the estimated generalization loss as calculated in step 5 above. However, one side effect of calculating $NN_k(p)$ for each fitted player p was that we could also reasonably model player p's stats during the predicted seasons by taking an weighted average of their neighbors' stats.

Shown below is an sample plot of a player, with several predicted and actual statistics graphed.



We considered four different models, each resulting in a different error or loss.

i. Truncated Model

The first model simply assumed that each player p would play no more seasons that we had already witnessed. This resulted in a surprisingly good loss of 1.2809, which can be attributed to the majority of NBA players playing very few seasons. Since we randomly selected how to truncate the player data, players with a lower amount of seasons played also had a lower error. Indeed, the distribution of these errors confirms this.

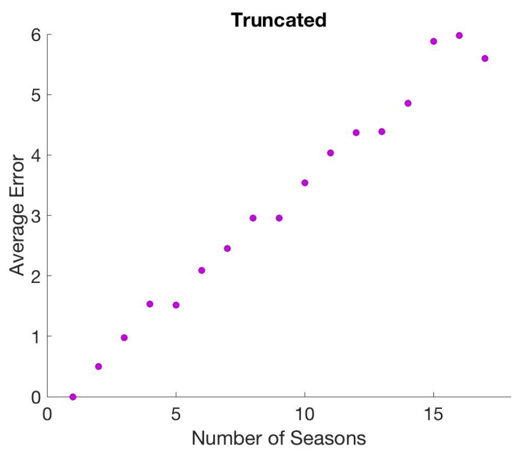


Figure 1: The average error for a player versus the number of seasons they played in a Truncated Model

ii. Uniform Feature Weighting Model

Our second model assumed that all features were weighted uniformly (θ is the matrix of 1s), so no coordinate descent was needed. With $k = 10$, this resulted in an estimated generalization loss of 2.9241, which was more than twice that of the naive model. However, the distribution of these losses is significantly different:

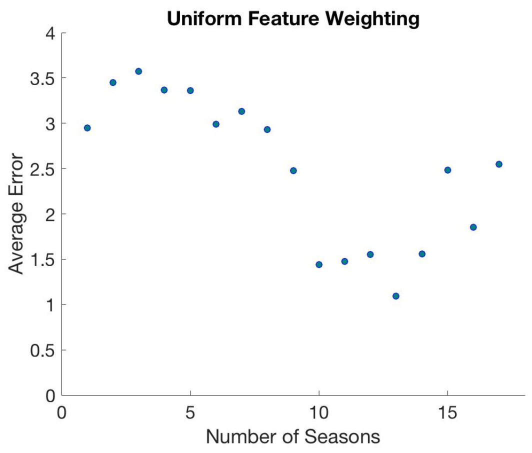


Figure 2: The average error for a player versus the number of seasons they played in a Uniform Feature Weighting Model

As can be seen from the scatter plot, this graph looks like a flipped version of the scatter plot for the truncated model! The disruption in the downward trending average error near the end of this graph can be explained by randomness and high variance. There are less than five players that played sixteen or more seasons, so the average error depends on how we randomly truncate and also how we split the data, and each player has a large impact on the average. However, it makes sense that our losses are largest for players who only play one or two seasons, and are therefore harder to fit. On the other hand, we are much more accurate on players that play a considerable number of seasons.

While this results in higher overall loss on the dataset, the loss distribution is arguably more desirable than the one in the truncated model. If we are accurate within two or three seasons for most players, then the salary range that we offer them is likely in the ballpark of what they are worth. However, if there is a significant probability of being off by 5 or more seasons, as is the case in the truncated model, then we grossly overpay a player, which is definitely something we wish to avoid.

In the future, we would like to explore other loss functions that may be more representative of a desirable model than our current one. One possibility is the loss function defined by the player distance between the predicted and actual players. Another possibility is the squared difference between the predicted and actual seasons played. Both of these punish larger losses more heavily than our current loss function.

iii. Tuning k, the number of neighbors

The realization that our current loss may not be the best indicator of a good model leads to complications when tuning k.

In fact, the lowest error for the uniform feature weighting model was 2.14 and occured when $k = 1$. This is because for many players, predicting future seasons based on a sole neighbor is very similar to our naive model of simply predicting no additional seasons, and as we saw before, this model has very low error. As k increases to 5, the error increases to 2.36, and as k increases to 10, the error increases even more to 2.9241. However, despite the increase in error, our distributions also become much more desirable with these higher errors.

Thus, instead of tuning k based on our current definition of loss, we tried tuning it with the loss defined as the player distance between the predicted player and the actual player. With this loss function, as k increases, we get better and better predictions, but with diminishing marginal returns. However, increasing k past a certain point (i.e. 50) was detrimental for players whom had played a significant number of seasons. This was because once the number of neighbors reaches a significant fraction of the total players, there would not be enough players who had played a significant number of seasons to fill up the neighbor set, and thus, we are likely to predict an end to their career that is far too early. Ultimately, we settled upon $k = 10$ as the marginal gains achieved by increasing k slightly were minimal.

iv. Learned θ values

Our last two models use learned θ values, with the third model containing all 20 original features, and the fourth model containing only 12 features. The features that were removed in the fourth model were features such as offensive and defensive rebounds (which sum to the rebounds feature that was left in the model), field goals made and field goals attempted (whose ratio is equal to the field goal percentage feature that was left in the model), and so on.

GAMES	1.0000	PF	-1.9867
MIN	1.3851	FGM	0.4660
POINTS	1.3657	FGA	0.9850
ORB	1.0935	FG%	0.1336
DRB	1.4365	3PTM	0.5182
REB	1.4066	3PTA	1.2226
AST	0.7679	3PT%	1.8686
STL	0.6929	FTM	1.9560
BLK	0.4321	FTA	2.0713
TO	-1.0876	FT%	1.0000

Figure 3: Learned theta values without any feature selection, estimated generalization loss: 1.875

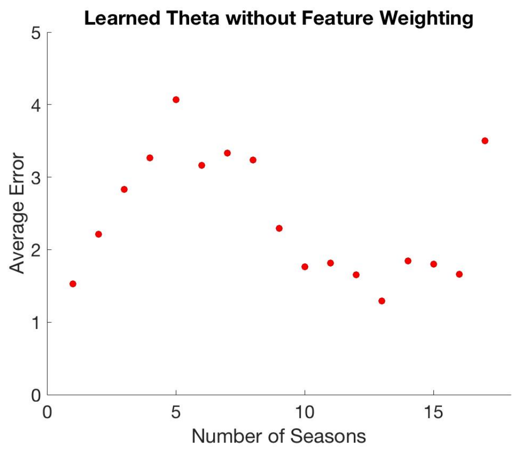


Figure 4: The average error for a player versus the number of seasons they played with a learned θ

The scatter plot of average errors for our learned θ model looks very similar to the one from our uniform feature weighting model. However, the learned θ model seems to have also discovered a way to accurately detect players that spend virtually no time in the league, something that the uniform feature weighting model was not able to do. It is no surprise then that the average error for all players was approximately 33% less than that of the uniform feature weighting model since these players constitute a significant portion of the NBA playerbase.

GAMES	1.0000	PF	2.0489
MIN	0.6422	FGM	-
POINTS	2.1250	FGA	-
ORB	-	FG%	0.2591
DRB	-	3PTM	-
REB	0.3524	3PTA	-
AST	1.8319	3PT%	1.1796
STL	-0.2860	FTM	-
BLK	1.4208	FTA	-
TO	-1.2048	FT%	0.8940

Figure 5: Learned theta values with feature selection, estimated generalization loss: 2.4375

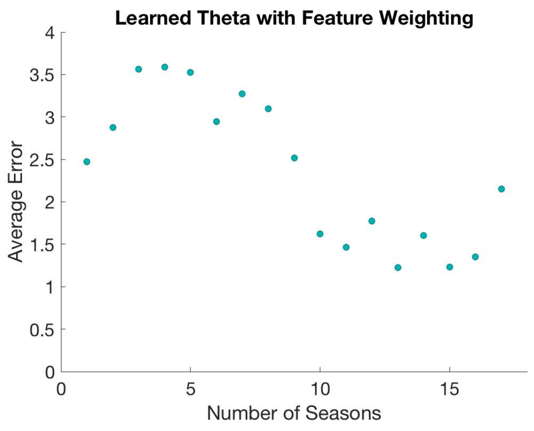


Figure 6: The average error for a player versus the number of seasons they played with a learned θ and feature selection

The scatter plot of average errors for our learned θ with feature selection model is virtually identical to the scatter plot for the uniform feature weighting model. This was surprising to us because we expected the feature selected model to perform at least as well as the the previous learned θ model, since the features we removed were all dependent on other still existant features. It is possible that our learned θ model is overfitting, which is why the error is so much lower than any of our other models. Another possible cause for the difference could just be the randomness in our coordinate descent algorithm. This could be tested by running coordinate descent multiple times with varying starting values. However, we did not have the time to do so because coordinate descent takes several days to run.

V. Future Work

The most important work to do in the future is to try different loss functions and find one that better represents our objective (minimize the risks that a NBA owner takes in signing a player to a new contract). Possibilities for loss functions include player distance as defined in our methods, and squared difference in number of seasons played.

We can also take a different approach to the problem. Instead of computing k-NN for a player p and performing coordinate descent, we could use k-means to cluster players from our training data, based on the types of players that exist in the NBA. We can then apply locally weighted regression to all players in the cluster to model player p's future. This would allow us to use stochastic gradient descent to learn θ.

To extend the training data available to our model, we may look into collecting player statistics from even farther back in history. However, the usefulness of this data is questionable since the game was very different back then, and therefore, the addition of this data may not be very relevant in predicting the career path of current and future players. Nevertheless, it would not hurt to decide whether or not more data is helpful after seeing the results.

References

[Gunday and Karasu, 2014] Gunday, G. and Karasu, A. (2014). NBA Miner Player Basic Stats. Retrieved November 10, 2016 from http://www.nbaminer.com/player-basic-stats/.