Correlating athlete value with statistics: a NBA case study

Brian Bao (btb79), Peter Li (pl488)

May 18, 2018

Abstract

Among the most lucrative of modern-day entertainment businesses, the NBA has grown into an entity worth over $30 billion, signing individual athletes to contracts worth as much as $20 million a year. As a study, we will explore the implications of performance statics on how NBA franchises value athletes. Our goal in this project is to apply data mining techniques to historical NBA data to predict a player's projected next-step valuation and identify the most critical performance stats (predictors) in doing so.

1 Introduction

In this paper, we will be analyzing time-series data for NBA player statistics and salary, which are both reported on a yearly basis. Our analysis will attempt to apply both supervised and unsupervised learning techniques to player stats to predict his salary for the following year. We plan to employ regression to predict the player's continuous salary, classification (SVM, logistic regression, decision trees, etc.) to predict the player's salary quartile, and unsupervised dimensional reduction methods like PCA to analyze importance of performance statistics on salary.

2 Model

Because we are predicting salaries, it makes the most sense to model this as a regression task. After experimenting with various machine learning regression algorithms, we also decided to try modelling it as a classification task. Thus we split the paper up into two primary sections: regression and classification. To model it is a classification task, we bin the observations into buckets. One primary reason for this is that the dataset might be significantly skewed for a certain year, or contain high outliers (e.g. Stephen Curry or LeBron James). Thus, to remedy this, we can pseudo-normalize the data by binning it into one of four categories, which is explained in more detail later.

3 Data preprocessing

3.1 Datasets used

Furthermore, since we aim to correlate next-step salary with performance statistics, we will need to draw data from multiple sources. First, we have the regular season (since regular season contains vastly more data points while) player stats dataset from Kaggle which contains all comprehensive player stats from 1990 to 2017. This dataset contains yearly statistics such as: total Points, total Assists, variations of Rebounds, Shooting Percentages, Personal Fouls, etc. From this dataset, we've curated a list of predictors that are reasonably independent to use as metrics for next-step salary prediction. Those predictors include:

- Total Assists (AST)
- Total Blocks (BLK)
- Total Defensive Rebounds (DREB)
- Total Offensive Rebounds (OREB)
- Free Throw Percentage (FT_P)
- Player Efficiency Rating (PER)
- Total Personal Fouls (PF)
- Total Points (PTS)
- Total Steals (STL)
- 3 Point Shot Percentage (3_P)
- 2 Point Shot Percentage (2_P)
- True Shooting Percentage (TS)

- Total Turnovers (TO)

Additionally, we have augmented the stats dataset using a player salaries dataset [1] from which we extracted player salaries by year played in the NBA. Any data points resulting in discrepancies between the salaries and the stats datasets resulted in exclusion of that data point. For instance, if Michael Jordan's 1995 season was included in the salaries dataset but not in the stats dataset, we ignored 1995 for Michael Jordan entirely. This was a key decision in cleaning the data since it left us with fully populated observation points for analysis on player years.
Preliminary results have revealed that

3.2 Data cleaning

To simplify the data, we decided to consider only players who have played sufficiently long careers in recent history. This means our models will consider only players who played 7 or more years between the season ending in 1990 and the season ending in 2018 (present day). This gives us about 7,390 player year observations. We note here that we did not include playoff statistics since playoff games played varies by year. In reality, this would be a major consideration on predicting salary since teams in the NBA tend to value athletes based on how far in the overall tournament they believe players will facilitate progress toward.

Finally, in tying up loose ends we noticed that the stats dataset sometimes has multiple athlete entries in a single year. This occurrence is indicative of an athlete moving teams in the middle of a season and recording qualifying performance stats throughout the year. We were able to reconcile this phenomenon by congregating the stats from all teams involved via appropriate methods: percentages were averaged using a weighted mean over games played for each team, and total counts were added up. We note here that this phenomenon does not occur in the salaries dataset since it is customary for a team receiving a player to buy out the player's existing contract.

Thus, salaries can be treated as if they were dispensed to the athlete on a yearly basis.

3.3 Data assumptions

Next, we note that we will decouple all player data from the team they were on since we are isolating performance statistics in our prediction and regression problems. This means that we will assume no major business decisions influenced player salary. Examples of such business decisions include but are not limited to: 1) player takes a pay cut to remain on a team for championship chasing purposes, 2) player changes teams to maximize salary opportunities, 3) any influence related to team affiliation, 4) penalties taken t salary as a result of public misconduct, etc.

3.4 Normalization

To achieve better performance given the varying scales of different features, we normalized the data to have zero mean and unit variance in each continuous feature by using min-max normalization, which normalizes according to the following:

$$x = \frac{x - min(x)}{max(x) - min(x)}$$

3.5 Bucketing

As mentioned above, in order to change the output value y from a regression value to a classification value, we bin the observations into 4 separate buckets. Each bucket represents the salary quartile for that year. For example, 0 represents the 1 to 25th percentile, 2 is the 26th to 50th percentile, 3 is the 51st to 75th percentile, and 4 is the 76th to 100th percentile. Below is a plot of the data — the top and bottom datasets are representative for regression and classification respectively.

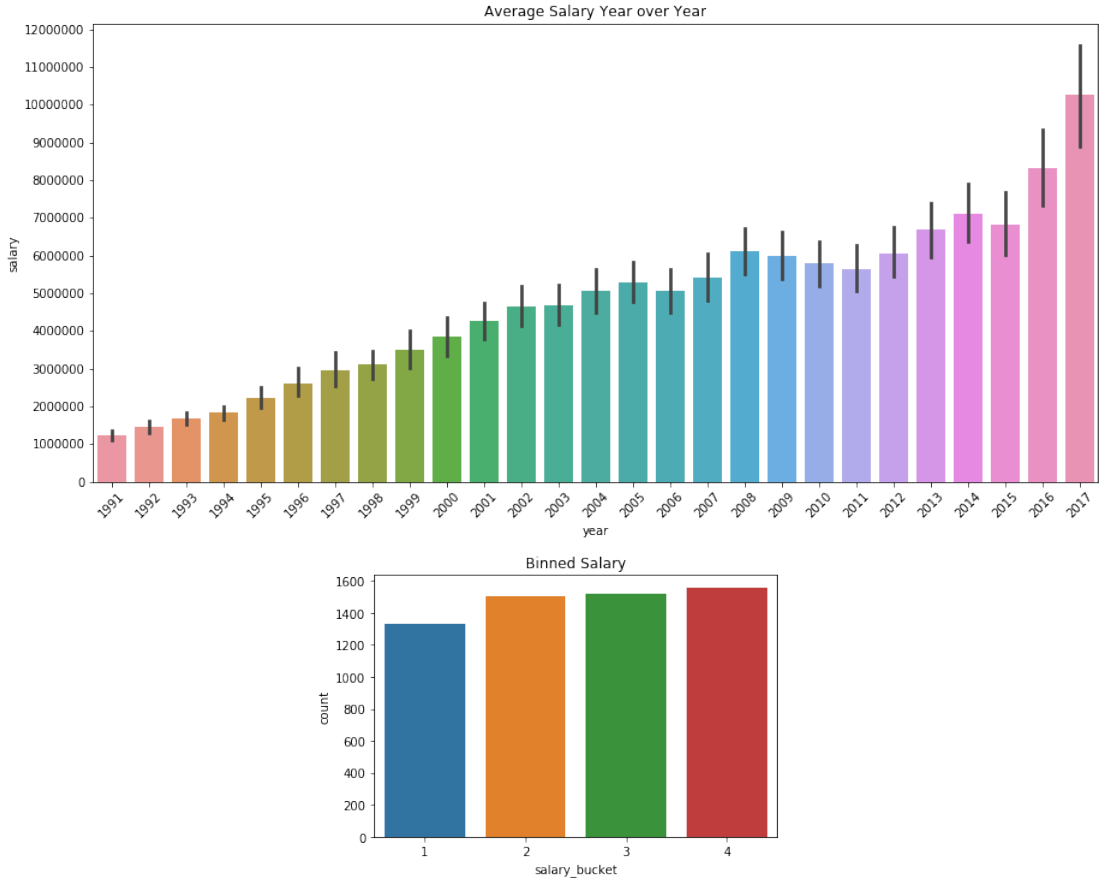

Figure 1: Before and after bucketing salary datapoints.

4 Feature plotting

Without applying any machine learning algorithms, we first plot the data on a high level to see if we can notice any overarching correlations with the feature set. Below is a sample of 4 features plotted against the count.

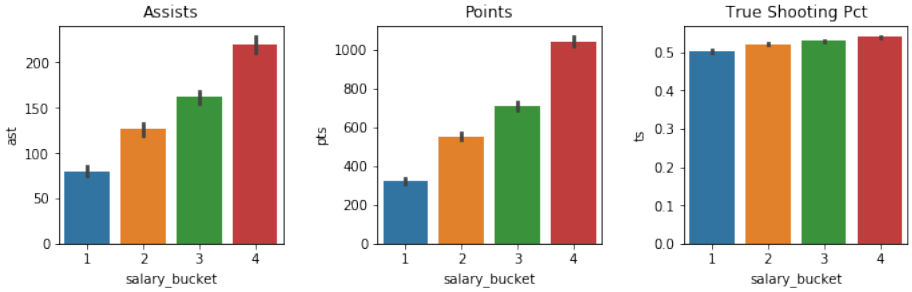

Figure 2: Plot of player statistics versus salary bucket

From the above plots, we can deduce that features such as assists and points are relatively strongly correlated with salary and true shooting percentage is a weak feature for salary. We will explore this more in detail later.

5 Classification and regression techniques

We used an array of different models to classify the data. Each method has different properties that make it better or worse in different types of datasets, so we were interested in seeing which would perform the best for our data.

5.1 k-nearest neighbors

The k-nearest neighbors classifier treats the space of features as a metric space and uses the entirety of the training data to classify testing points. Given a test point, we find its k nearest neighbors using a prescribed distance function (usually Euclidean distance), and the label we assign is the mode of the neighbors' labels. kNN generally produces decent results given the assumption that "similar" points will have the same label, which should hold in our case of predicting salaries based on player statistics.

Parameters:

- k: the number of neighbors

5.2 Linear and logistic regression

In short, linear and logistic regression are ways to fit a curve or hyperplane to the data in order to predict future observations. In particular, logistic regression is best used for classification, as we are computing $P(Y|X, D)$, or the probability that the label is y given that we have the training data x and some data distribution θ. For purposes of this project, we use linear regression for predicting the continuous salary, and logistic regression for predicting the salary bucket.

Parameters:

None.

5.3 Support vector machine (SVM)

Traditional SVM performs binary classification with a linear decision boundary, using a subset of the training points as "support vectors" that determine the classification and a regularizer that negotiates the tradeoff between training accuracy and complexity of the model (which can lead to overfitting). With some modifications though, we can get past both the linear and binary limitations. Using the kernel trick, we can bypass the linear boundary restriction and classification restriction. By using kernels to transform the data, we can model the data as a combination of the training points to fit the hyperplane.

Parameters:

- C: tradeoff for misclassifying a point
- kernel_type: type of kernel (e.g. polynomial, linear, or rbf in our case)
- γ: the amount of influence of a training support

5.4 Decision tree

Decision trees are predictive models that split the feature space on feature values until the resulting subset has the same label. Assuming the feature space is small enough, the decision tree should be able to achieve high accuracy as we can continuously split an impure node until it reaches full purity. Purity in this case is the fraction of observations with the same label.

Parameters:

- max_depth: the maximum depth of the tree
- max_features: the maximum number of features to consider when looking for the best split

5.5 Random forest

Random forest is an extension on bagged decision trees. Decision trees are extremely prone to high bias and overfitting since they hone in on certain features to split on. Thus, we use bagging to fix this issue. Bagging is the process by which we randomly select $n' = n$ observations with replacement from the training set to train a specific tree t_i. We then train k trees and then for a testing point x_{Te}, we take the average result for regression and majority vote for classification. Random forest is extremely strong in practice, as it reduces bias and decision tree overfitting by bagging and randomly selecting features to split on. In addition, it naturally engineers features via the purity metric defined above. However, one downside to random forests is that they become slow to train proportionately to the number of trees in the forest. As with any machine learning algorithm, they are also prone to overfitting in poor datasets.

Parameters:

- num_trees: the number of trees (estimators) in the forest
- max_depth: the maximum depth of any tree
- max_features: the maximum number of features to consider when looking for the best split

6 PCA

One thing we should always check is whether performing PCA on the dataset will help reduce the dimensionality at a low cost of error. First, we perform PCA on the dataset and plot the cumulative explained variance against the number of principal components. Recall that the first principal component is the strongest, the second is the second strongest, so on and so forth.

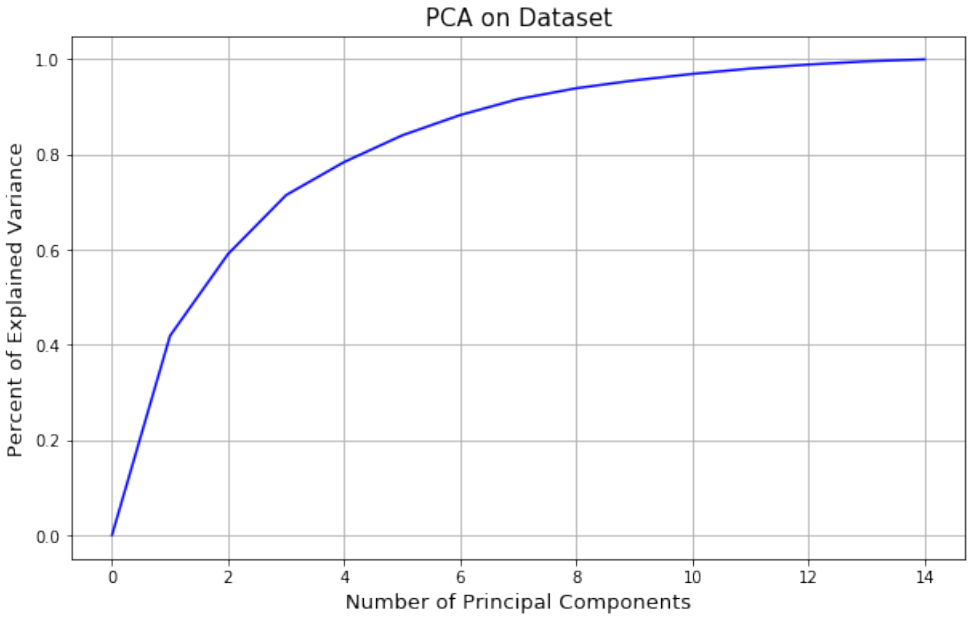

Figure 3: Cumulative Sum of Explained Variance per Principal Component

As evidenced by the above graph, 8 principal components explain more than 90% of the data's variance. Thus, we can cut the feature space almost in half. To further this argument, we plot the training and testing error as we use i principal components on the random forest algorithm.

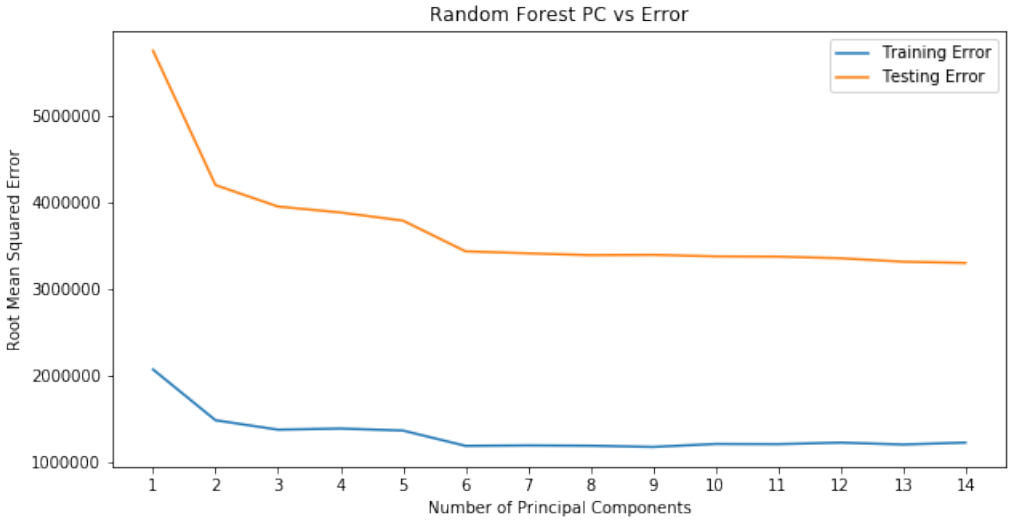

Figure 4: Plot of Root Mean Squared Error vs. # of Principal Components

The RMSE plateaus strongly after the 6th and 7th principal component. Hence, we

should reduce the dimensionality according to the first 8 principal components.

7 Hyperparameter tuning

To robustly find the optimal hyperparameters for our algorithms, we used grid-search k-fold cross-validation. Given an algorithm with a set of parameters and a set of pre-chosen values at different scales for each parameter, we perform k-fold cross validation with every combination of parameter values and picked the combination that gave the highest cross-validation accuracy. For our data, we used $k = 5$ and performed parameter tuning on the k-nearest neighbors, SVM, and random forest algorithms. There were no parameters to tune for linear and logistic regression, and we decided to keep the default max_depth value for the decision tree and random forest classifiers.

8 Results

To justify the more complex models, we've provided results from both: 1) a dummy regressor that guesses the median salary seen in training for the unsupervised continuous salary prediction, and 2) a dummy classifier that guesses the most common label seen in training for the supervised salary bucket prediction.

In addition, for some machine learning algorithms, we include their default parameters to indicate the strength of using grid search cross validation to tune their hyperparameters. Here are the full results of regression with and without PCA, and classification:

	baseline dummy	default knn	linear/logistic regression	default svm	decision tree	default random forest	tuned knn	tuned svm	tuned random forest
regression w/o pca	5.37m	3.65m	3.65m	5.37m	4.50m	3.48m	3.6m	4.06m	**3.22m**
regression w/ pca	5.37m	3.67m	3.76m	5.37m	4.63m	3.57m	3.63m	4.06m	**3.38m**
classification	26%	43%	46%	46%	34%	44%	44%	48%	**49%**

Figure 5: Results Table

9 Analysis

We can see from the results table that the tuned random forest performed the best. We can expect this to happen as random forests are very strong as they are an ensemble of bagged decision trees. Below we go into some analysis of the above:

9.1 Feature importance

As previously stated, the great thing about random forests is that they have the natural ability to generate how good a specific feature is at splitting the dataset by using the impurity metric. Below is the plot for regression on all features.

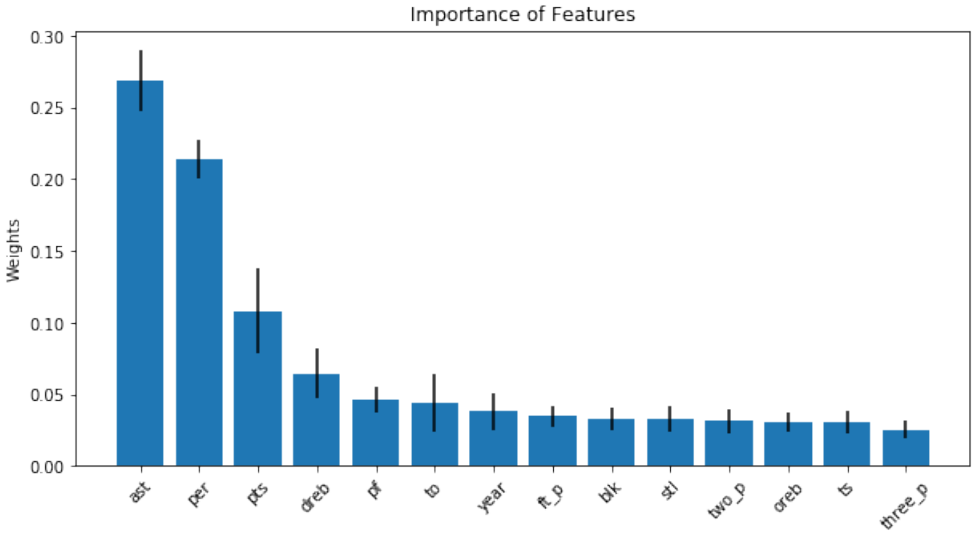

Figure 6: Plot of Feature Importance According to Their Impurity

Recall from earlier that we predicted True Shooting Percentage (TS) would be a poor evaluator and Assits would be a strong one. The above plot confirms this.

Furthermore, the strongest predictors: Assists, Player Efficiency Rating, Points, and Defensive Rebounds appear to be within expectations. We can infer that high values in these statistics are commonly observed in players who play at the Point Guard or Shooting Guard positions, which are currently the highest paid positions in the NBA on average.

9.2 Tuning hyperparameters

Tuning hyperparameters was very powerful. In all the datasets, tuning the hyperparameters within a small hyperparameter subspace gave a significant increase to accuracy. However, the downside of tuning hyperparameters is that the runtime grows exceedingly quick as the number of parameters grows linearly. Therefore, it is very easy to get stuck at a local optimum by telescopic searching into a particular hyperparameter interval.

10 Conclusion

Overall, we achieved a relatively good accuracy in comparison to the baseline dummy classifier.

There are also other modifications that can be explored within the realm of our classification model. For example, there are many other types of classifiers that can be used. One machine learning algorithm we did not test was gradient boosting trees (using XGBoost), which can model subtle complexities and relationships in data that many other approaches cannot. Another possibility could be to group the data differently, such as grouping the data into only 2 buckets (splitting into simply "high" and "low") and treating it as a binary classification problem. A final area of exploration is the features themselves. For example, an interesting feature we could consider is to add the name of the player as a categorical feature, using a "one-hot" vector to represent the name of the player.

References

[1] https://data.world/datadavis/nba-salaries/workspace/file?filename=nba_salaries_1990_to_2018.csv