NBA Salary Prediction

Brandon Arcilla

Problem

NBA Free Agency can be an exciting time for teams, players, and fans. Players want to know which teams want them and teams want to know which players can help. An important factor for both parties is determining salary.I want to be able to predict a salary range for players based off advanced NBA statistics. NBA teams could use this to help gauge how much a player should be making or which players fit into their salary situation. An NBA Player could use this to see what they could potentially be earning based off of their production on the court.

Data Wrangling

Description of data obtained:

Data	**Description**
Player Name	The name of the NBA player
Player ID	The unique identifier used by the website
Pos	NBA Position: Point Guard (PG), Shooting Guard (SG), Small Forward (SF), Power Forward (PF), Center (C)
Age	Age of Player at the start of February 1st of that season.
Tm	Team that the NBA player belonged to
G	Total # of games played
MP	Total # of minutes played
PER	Player Efficiency Rating: A measure of per-minute production standardized such that the league average is 15.
TS%	True Shooting Percentage: A measure of shooting efficiency that takes into account 2-point field goals, 3-point field goals, and free throws.
3PAr	3-Point Attempt Rate: Percentage of FG Attempts from 3-Point Range
FTr	Free Throw Attempt Rate: Number of FT Attempts Per FG Attempt
ORB%	Offensive Rebound Percentage:An estimate of the percentage of available offensive rebounds a player grabbed while he was on the floor.

DRB%	Defensive Rebound Percentage: An estimate of the percentage of available defensive rebounds a player grabbed while he was on the floor.
TRB%	Total Rebound Percentage: An estimate of the percentage of available rebounds a player grabbed while he was on the floor.
AST%	Assist Percentage: An estimate of the percentage of teammate field goals a player assisted while he was on the floor.
STL%	Steal Percentage: An estimate of the percentage of opponent possessions that end with a steal by the player while he was on the floor.
BLK%	Block Percentage: An estimate of the percentage of opponent two-point field goal attempts blocked by the player while he was on the floor.
TOV%	Turnover Percentage: An estimate of turnovers committed per 100 plays.
USG%	Usage Percentage: An estimate of the percentage of team plays used by a player while he was on the floor.
OWS	Offensive Win Shares: An estimate of the number of wins contributed by a player due to his offense.
DWS	Defensive Win Shares: An estimate of the number of wins contributed by a player due to his defense.
WS	Win Shares: An estimate of the number of wins contributed by a player.
WS/48	Win Shares Per 48 Minutes: An estimate of the number of wins contributed by a player per 48 minutes (league average is approximately .100)
OBPM	Offensive Box Plus/Minus: A box score estimate of the offensive points per 100 possessions a player contributed above a league-average player, translated to an average team.
DBPM	Defensive Box Plus/Minus: A box score estimate of the defensive points per 100 possessions a player contributed above a league-average player, translated to an average team.
BPM	Box Plus/Minus: A box score estimate of the points per 100 possessions a player contributed above a league-average player, translated to an average team.
Salary	Salary for player during that year
clean_Salary	Processed salary removing decimal and $
%_of_cap	Percentage of clean_Salary against the team salary cap

Season	Year that was played

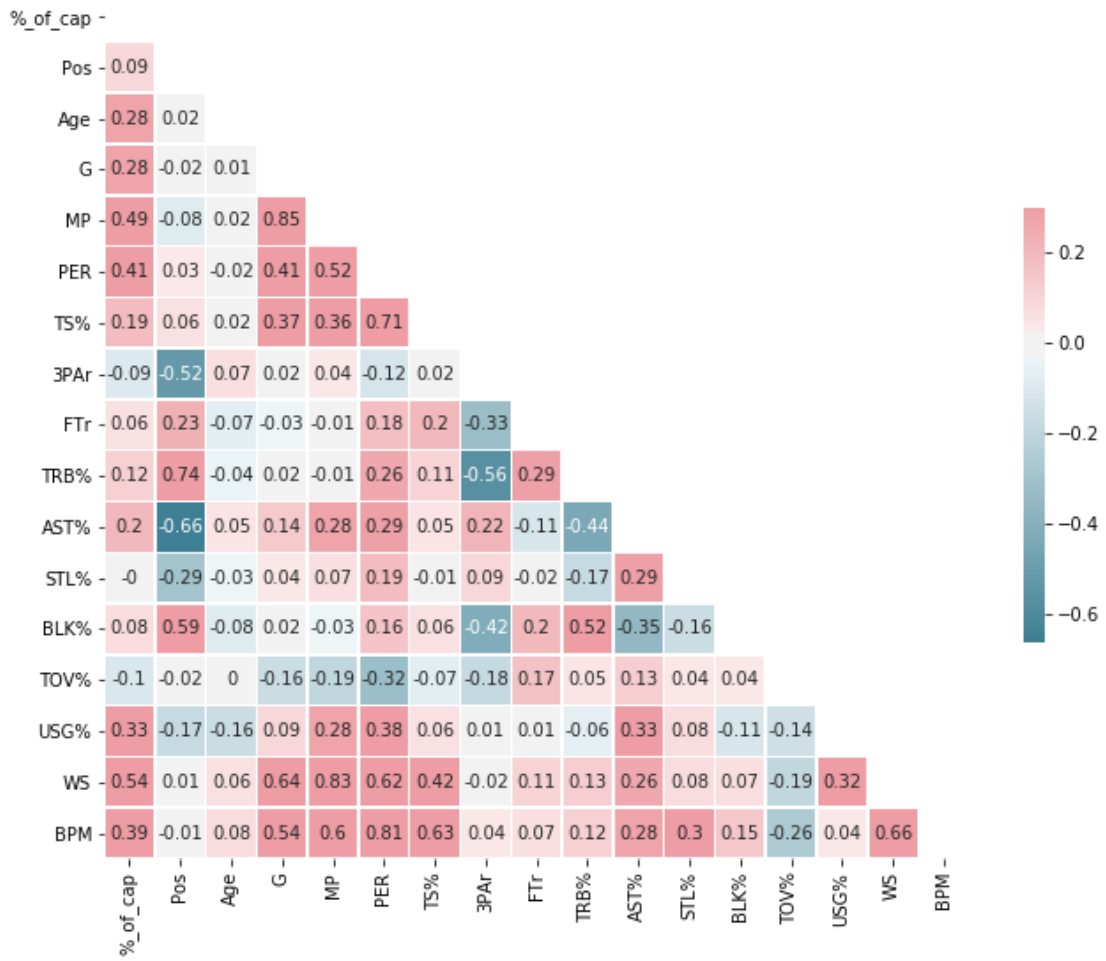
- Data sources
    - NBA Salary (1985-2017) from data.world: This data set contained a player_id, which acted as the unique identifier. This dataset included salary and year for each player. Not much cleaning was done on this data set.
    - Team Salary Cap (1997-2017) from data.world: Since inflation is not taken into account for player salary this data is used to normalize the players salary to a percentage of the salary cap for each year. Team Salary Cap is the maximum amount of money a team could use on their roster. It should be noted that there are exceptions where teams can exceed the cap amount, either by exceptions based on player contract or by paying a luxury tax. This type of data is not being considered in this project.
    - NBA Advanced Stats (1997-2017) from www.basketball-reference.com: BeautifulSoup was used to scrape the website. The following steps were done to wrangle and cleanse the data.
        - To get a list of urls to scrape, I first identified that each url is the same with the exception of the year.
        - A list of years from 1997 to 2017 was created.
        - Then the URL was iterated with each year to get a list of URLs.
        - A function was defined for the scraping (urlScraping).
            - Within this function, BeautifulSoup is used to open the url.
            - Column headers, player_stats, player_id, and year were extracted.
            - A DataFrame is created using headers as columns and player_stats as rows.
            - There were blank columns in the DataFrame that were taken out using isspace().
            - There were also rows with all NaN values that were removed as well.
            - Extracted data player_id and year were added as columns.
            - DataFrame index was reset, then the index column was dropped.
            - Certain columns were converted from Object to Float/Int.
        - A for loop was created to run all url through urlScraping function to save as a csv with matching year.
        - Another for loop was used to read_csv and append all data into an empty list where that list was concatenated to create one list of advanced stats for players from 1997-2017.

Final output of the dataset was done by merging advanced stats to the players salary on the player_id and year. Left join was used because I wanted to see how many players didn't have

any salary in the salary dataset. Players without salary accounted for less than 5% of the total data, so they were excluded.

Exploratory Data Analysis

Heatmap of Correlation



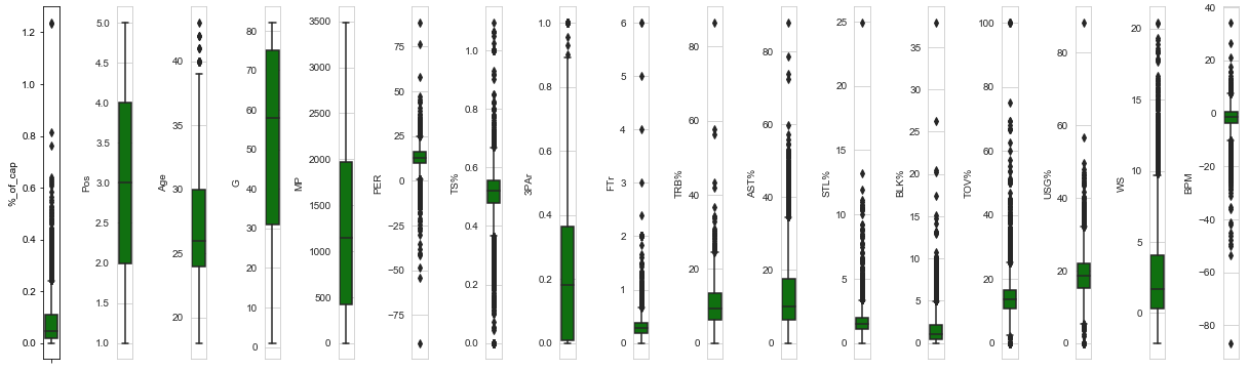
Looking at the correlation between % of Salary Cap against other variables, it can be broken down as:

Moderate Positive Correlation (0.3 to 0.7)	Minutes Played, Player Efficiency Rating, Usage %, Win Shares, Box Plus/Minus

Weak Positive Correlation (0 < c ≤ 0.3)	Position, Age, Games Played, True Shooting %, Free Throw Rate, Total Rebounding %, Assist %, Block %
Weak Negative Correlation (-0.3 ≤ c < 0)	3-Point Attempt Rate, Turnover %
No Correlation (0)	Steal %

What surprises me the most is that the variables that I believed would be the strongest correlated to % of salary cap actually has a weak positive correlation. It would seem how many minutes played, efficiency rating, usage percentage, win shares, and box plus/minus are more telling for a higher salary %

Outliers

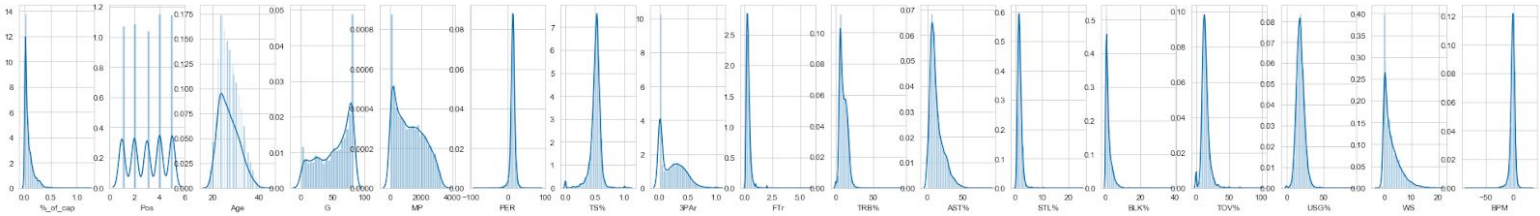


Position, Games Played, Minuted Played are the only variables that do not have outliers. This makes sense because there are no other positions, games are set to a certain amount and minutes played per game is set too.

The outlier for % of salary cap goes to Michael Jordan during the 96-97 and 97-98 NBA Season. During this year there was no maximum salary cap for players. He was making more than the salary cap allowed.

Player	Team	Season	Salary	Salary Cap	%_of_cap
Michael Jordan*	Chicago Bulls	1997-98	33140000	26900000	1.231970
Michael Jordan*	Chicago Bulls	1996-97	30140000	24363000	1.237122

Data Distribution



It looks like the variables are skewed to the right. This makes sense as these variables would never be below 0, except for BPM (Box Plus/Minus).

Data Preprocessing

Here is what the original data looked like.

	%_of_cap	Pos	Age	G	MP	PER	TS%	3PAr	FTr	TRB%	AST%	STL%	BLK%	TOV%	USG%	WS	BPM
0	0.113043	1	26	57	2029	18.6	0.535	0.324	0.164	3.9	34.7	1.6	0.1	10.1	25.1	5.5	1.1
1	0.015652	3	31	53	516	9.6	0.489	0.055	0.133	10.6	9.2	0.9	1.3	13.4	17.7	0.2	-5.2
2	0.021304	1	22	60	560	8.7	0.506	0.426	0.161	4.2	31.6	2.5	0.3	29.1	18.8	0.2	-4.7
3	0.008696	2	23	41	362	7.8	0.447	0.314	0.343	4.1	20.8	3.0	1.1	22.0	19.0	-0.3	-6.1
4	0.015217	5	23	73	1614	11.8	0.518	0.008	0.289	10.8	6.5	1.8	2.4	11.9	13.7	1.7	-1.8

Some preprocessing that I needed to do was to bin '%_of_cap' by ranges in order to make this a classification problem and to one-hot encode my categorical feature 'Pos'. I used pandas get_dummies to one-hot encode my categorical feature. To group my data, I used NBA minimum (10%) and maximum (25%) of salary cap allowed to obtain my first and last groups, then segmented what was in between.Finally I removed the %_of_cap and Pos column, and split my data into feature and target variables. The data was rescaled using MinMaxScaler.

Machine Learning Models

5 Models were used: Decision Tree, Random Forest, XGBoost, AdaBoost and SVM.

For the model selection process, I created a pipeline that combines the preprocessing MinMaxScaler with a classifier dictionary containing all the models selected with default parameters. I looped the pipeline with KFold cross-validation to measure model performance.

	classifier	mean_fit_time	mean_test_score	std_test_score	mean_train_score
0	DecisionTree	0.095745	0.666398	0.007840	1.000000
1	RandomForest	0.174659	0.745540	0.009935	0.983501
2	SVC	1.227508	0.730651	0.011307	0.731388
3	AdaBoost	0.874373	0.747150	0.009784	0.758417
4	XGBoost	5.135772	0.758417	0.011314	0.793427

The following metrics were used to evaluate our models:

1. Average test score
    a. The average test score is our accuracy score. This tells us which model performed well on the test data. Decision Tree scored the lowest at 0.667. The other 4 classifiers were very similar with Random Forest edging out as the top model performer in test scores.
2. Average train score
    a. The average train score tells us how well the model performed on trained data set. When comparing the train score and the test score, we can get an idea if our model overfits on the training data, that is it predicts very well on the train data but not as well on the test data, by looking at the difference between train scores and test scores. The larger the difference, the more your model overfits. We can that Decision Tree and Random Forest Classifier models overfit on the training data by a larger margin than the other models.
3. Test score standard deviation
    a. The standard deviation of the test score lets us know the variance of our models. We are looking for models that have a low variance, which will tell us if the prediction we obtained on test is not by chance and that our model will perform similarly on all test sets. SVC and XGBoost have the largest variance.
4. Average fit time
    a. This metric is the average time, in seconds, it takes to fit the model to the data. I want my model to run fairly quick. XGBoost and SVC take the longest to fit, while Decision Tree is the fastest.

The models that performed well enough to dive deeper into were AdaBoost and Random Forest Classifier. AdaBoost Classifier had a high accuracy score, did not overfit, had a low variance, but it does take longer to fit. Random Forest Classifier had a high accuracy score and takes less time to fit but the model does overfit on the training model.

Hyperparameter Tuning

I used GridSeachCV to find optimal parameters.

AdaBoost Tuning:

The hyperparameters that were tuned were learning_rate and n_estimators. The best estimator used 0.1 for the learning rate and 1000 for n_estimators. This lead to an improvement in accuracy score at 0.764

```
ada_predict = pipe_ada.predict(X_test)
print('Model Accuracy: ', accuracy_score(y_test, ada_predict))
print('Training Score: ' , pipe_ada.score(X_train, y_train))
print('Best Parameters: ', pipe_ada.steps[1][1].best_params_)
```

```
Model Accuracy:  0.7637826961770624
Training Score:  0.7691482226693495
Best Parameters:  {'learning_rate': 0.1, 'n_estimators': 1000, 'random_state': 34}
```

```
print(confusion_matrix(y_test, ada_predict))
```

```
[[1762   33    2   11   14]
 [ 192   39    8    4   15]
 [  84   38   11   10   28]
 [  36   15    9   14   22]
 [  18   23    6   19   72]]
```

```
print(classification_report(y_test, ada_predict))
```

```
              precision    recall  f1-score   support

       0-10%       0.84      0.97      0.90      1822
     10%-15%       0.26      0.15      0.19       258
     15%-20%       0.31      0.06      0.11       171
     20%-25%       0.24      0.15      0.18        96
       25%<=       0.48      0.52      0.50       138

   micro avg       0.76      0.76      0.76      2485
   macro avg       0.43      0.37      0.38      2485
weighted avg       0.70      0.76      0.72      2485
```

Random Forest Tuning:

In this classifier I tuned max_features, n_estimators, and max_depth to help counteract the model overfit. The best parameters found was ‘auto’ for max_features, 2000 for n_estimators, and 8 for max_depth. The model accuracy improved to 0.763 and brought the training score down to 0.795. That’s a significant improvement from the default method.

```
rf_predict = pipe_rf.predict(X_test)

print('Model Accuracy: ', accuracy_score(y_test, rf_predict))
print('Training Score: ' , pipe_rf.score(X_train, y_train))
print('Best Parameters: ', pipe_rf.steps[1][1].best_params_)
```

```
Model Accuracy:  0.7625754527162978
Training Score:  0.794634473507713
Best Parameters:  {'max_depth': 8, 'max_features': 'auto', 'n_estimators': 2000, 'random_state': 34}
```

```
print(confusion_matrix(y_test, rf_predict))
```

```
[[1803    6    1    1   11]
 [ 226   16    5    0   11]
 [ 126   10    4    1   30]
 [  63    5    0    0   28]
 [  50   10    5    1   72]]
```

```
print(classification_report(y_test, rf_predict))
```

```
              precision    recall  f1-score   support

       0-10%       0.79      0.99      0.88      1822
     10%-15%       0.34      0.06      0.10       258
     15%-20%       0.27      0.02      0.04       171
     20%-25%       0.00      0.00      0.00        96
       25%<=       0.47      0.52      0.50       138

   micro avg       0.76      0.76      0.76      2485
   macro avg       0.38      0.32      0.31      2485
weighted avg       0.66      0.76      0.69      2485
```

In this situation, I want the end user to be able to see who is getting over and undervalued for the kind of stats they bring into a game. For an NBA teams front office, potentially finding players who can give the kind of stats that warrant <20% of the salary cap but currently getting paid lower and vice versa. Or for NBA players to gauge, with their current stats, whether they are getting categorized at a higher or lower percentage of the salary cap than what they currently are being paid. For this reason, I placed more importance on recall value. AdaBoost Classifier had the lower variance and higher recall and precision values

Model Analysis

In [897]:
```
features_ada = pd.DataFrame()
features_ada['feature'] = feature_names
features_ada['weight'] = pipe_ada.steps[1][1].best_estimator_.feature_importances_
features_ada.set_index('feature').sort_values('weight', ascending=False)
```

Out[897]:

feature	weight
MP	0.162
G	0.139
Age	0.117
TS%	0.078
TRB%	0.078
BPM	0.067
USG%	0.066
FTr	0.054
TOV%	0.050
PER	0.039

Feature importance of AdaBoost Classifier show that Minutes Played, Games Played and Age are the top features. This makes sense because the more you play, in games and minutes, means that the value you bring to that team is high enough for them to always play you. This can correlate to a higher salary. Age is also very telling because the younger you are, the more likely you are to either be a rookie or still be on a rookie contract. A rookie may play phenomenally for their team but their salary is capped since they are a rookie.

Conclusion and Future Work

In this project, we predicted salary categories for NBA players based on basic player demographics and advanced NBA statistics. Various machine learning models were used, with AdaBoost Classifier performing the best. The classifier was able to predict NBA players salaries with an accuracy score of 76.4% and recall score for '20-25% group' and '<=25% group' at 15% and 52%, respectively.

In the future, I would add more features that involved more than just what is being done on the basketball court, such as years of experience, what kind of contracts are the players currently and/or previously on and how long were the players on these contracts.