ORIE 4741 Final Report: Predicting NBA Salaries

Genghis Shyy (gs484) , Heesun Chang (hc483), Mohammad Kamil (mk848)

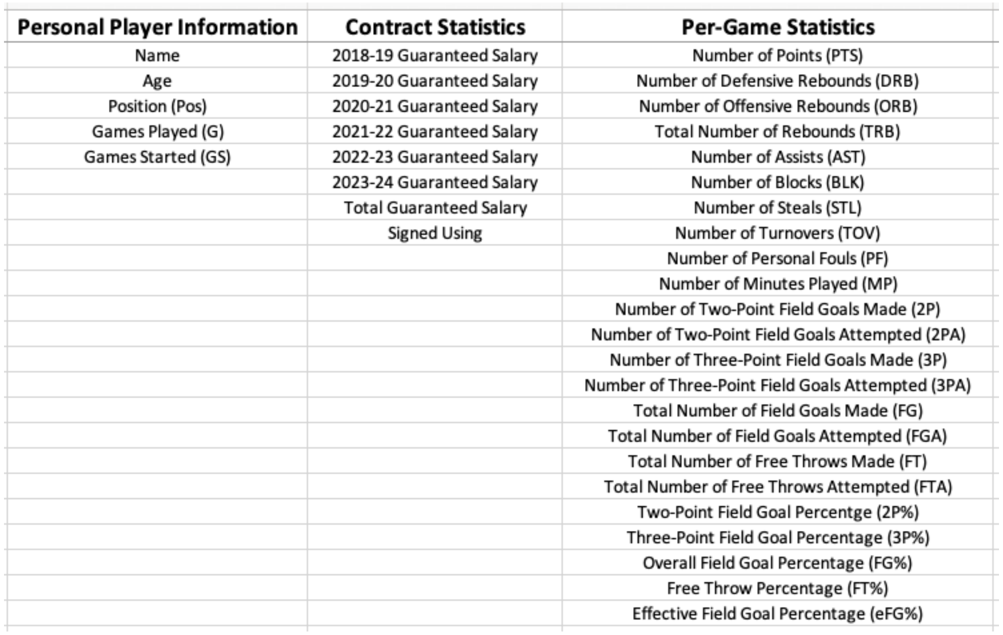
December 2020

1 Introduction

The goal of this project is to analyze the different relationships between NBA players' statistical performance and the contracts they receive. More specifically, we plan on exploring a number of questions such as: How much does a player's shooting efficiency typically influence the general perception of a player's points-per-game statistic? How does a player's position and/or age affect the statistical expectations that player receives? Can we even confidently assert that there does indeed exist a positive correlation between a player's points-per-game statistic and that player's salary or contract length? By exploring such potential relationships between players' statistical profiles and their contracts, we can ultimately hope to reliably predict the standard market contract any given NBA player should be awarded.

2 Dataset Overview

Overall, our preprocessed dataset focuses on 415 different NBA players, providing 1) all major 2017-18 regular season statistics, and 2) all contractual data over the 2018-19 through 2023-24 seasons for each player. More specifically, our dataset contains 36 distinct features for each player, such as points per game, total rebounds per game, assists per game, field goal percentage, total guaranteed salary, etc. We list all features in the table below:



Personal Player Information	Contract Statistics	Per-Game Statistics
Name	2018-19 Guaranteed Salary	Number of Points (PTS)
Age	2019-20 Guaranteed Salary	Number of Defensive Rebounds (DRB)
Position (Pos)	2020-21 Guaranteed Salary	Number of Offensive Rebounds (ORB)
Games Played (G)	2021-22 Guaranteed Salary	Total Number of Rebounds (TRB)
Games Started (GS)	2022-23 Guaranteed Salary	Number of Assists (AST)
	2023-24 Guaranteed Salary	Number of Blocks (BLK)
	Total Guaranteed Salary	Number of Steals (STL)
	Signed Using	Number of Turnovers (TOV)
		Number of Personal Fouls (PF)
		Number of Minutes Played (MP)
		Number of Two-Point Field Goals Made (2P)
		Number of Two-Point Field Goals Attempted (2PA)
		Number of Three-Point Field Goals Made (3P)
		Number of Three-Point Field Goals Attempted (3PA)
		Total Number of Field Goals Made (FG)
		Total Number of Field Goals Attempted (FGA)
		Total Number of Free Throws Made (FT)
		Total Number of Free Throws Attempted (FTA)
		Two-Point Field Goal Percentge (2P%)
		Three-Point Field Goal Percentage (3P%)
		Overall Field Goal Percentage (FG%)
		Free Throw Percentage (FT%)
		Effective Field Goal Percentage (eFG%)

Figure 1: Dataset features

This particular set of features not only provides detailed contractual information, but also ensures we can flexibly measure a player's statistical performance in a variety of ways, such as by shooting efficiency (by looking at a player's different field goal percentages); defensive impact (by looking at steals and blocks per game); ball-handling ability (by computing players' assist-to-turnover ratios), etc. Additionally, all missing or corrupted data was removed or fixed during preprocessing, which involved 1) aggregating all statistics for players who played for multiple teams during the 2017-18 NBA regular season; 2) combining contract salaries for players who received either multiple contracts or multi-year contracts so as to obtain these players' total guaranteed and non-guaranteed salaries; and 3) handling proper type conversions to avoid unexpected floating point errors, misplaced empty strings, etc. All data was originally found on `basketball-reference.com`, a third-party website created by a team of engineers, statisticians, sport analysts, and computer scientists. (The raw data itself is made available by National Basketball Association and its staff, who collects and stores relevant information after every game.)

3 Perceptron

To begin with, we utilized the perceptron algorithm in an effort to explore whether our data was linearly separable, or at least close to being linearly separable. Specifically, for each NBA season for which we have contract information (i.e., for each season from 2018-19 to 2023-24, inclusive), we plotted points per game ('PTS') vs. efficient field goal percentage ('eFG%') for every player that has a contract during that season. Each point on the plot was then categorized as $+1$ if its corresponding player earned at least the taxpayer's mid-level exception and -1 otherwise. (Note: The taxpayer's mid-level exception is a certain amount of money that is set annually by the NBA and is often utilized as a baseline approximation of the average NBA salary. This exception was set at \$8,600,000 for contracts starting in the 2018-19 NBA season; therefore, we utilized this value of \$8,600,000 throughout the rest of our study.) Classifying all data points in this fashion then allowed us to test whether our data is linearly separable—as linear separability here would imply a player's points per game and efficient field goal percentage can already be considered sufficient information to predict whether a player's salary meets the average salary baseline. We began with the 2018-19 NBA season, running the perceptron algorithm with a maximum of 1000 iterations to see if the algorithm would converge (Figure 2).

To the right, all blue points represent players who had a total guaranteed salary of at least the taxpayer's mid-level exception, while all magenta points represent players who had a total guaranteed salary less than the taxpayer's mid-level exception. Clearly, we can tell just from the plot alone that even increasing the maximum number of iterations would not cause the perceptron algorithm to converge, since this data for the 2018-19 NBA season is not linearly separable. Similarly, we found that the perceptron algorithm failed to converge after 1000 iterations for data on the 2019-20, 2020-21, and 2021-22 NBA seasons, since in each of these instances the data was evidently not linearly separable; and although the perceptron algorithm did converge for the 2022-23 and 2023-24 NBA seasons, there was only contractual data for only 16 and 3 players in these two seasons, respectively. The small sample size in both of these seasons effectively discouraged us from drawing any firm conclusions from this.

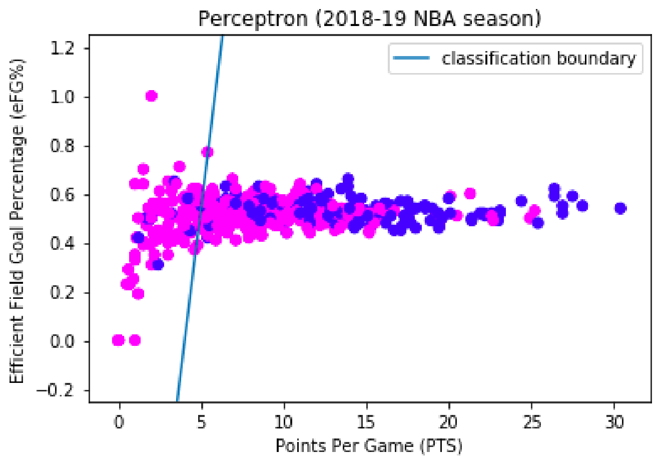


Figure 2: Running the perceptron algorithm on 2018-19 season data

Instead, we argue that the failure of the perceptron algorithm to converge on data for any of the other earlier seasons convincingly indicates that our data is not linearly separable. This in turn implies that we cannot simply rely upon points per game and efficient field goal percentage to reliably predict whether a player will earn a contract at least as high as the average salary threshold discussed above. To confirm this pattern extends beyond the case of these two specific features, we test a few other cases below, using the 2018-19 season data since it accounts for the highest number of players:

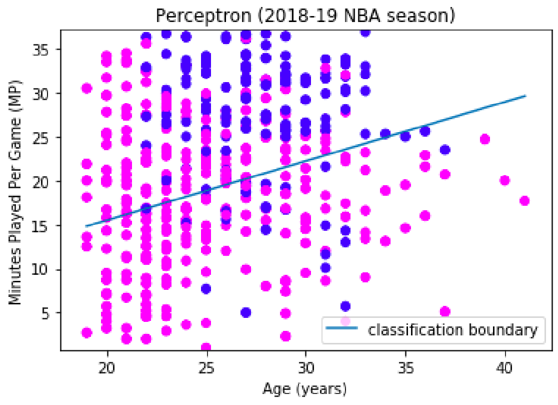


Figure 3: Running the perceptron algorithm when considering player age and minutes played per game

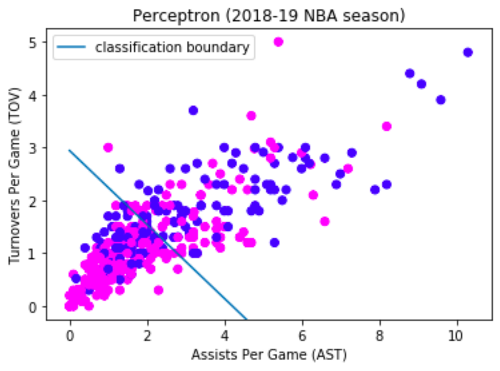


Figure 4: Running the perceptron algorithm when considering assists and turnovers per game

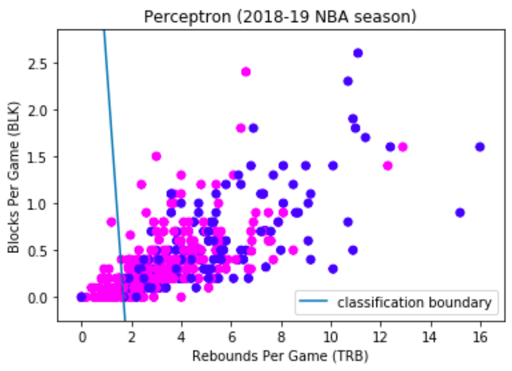
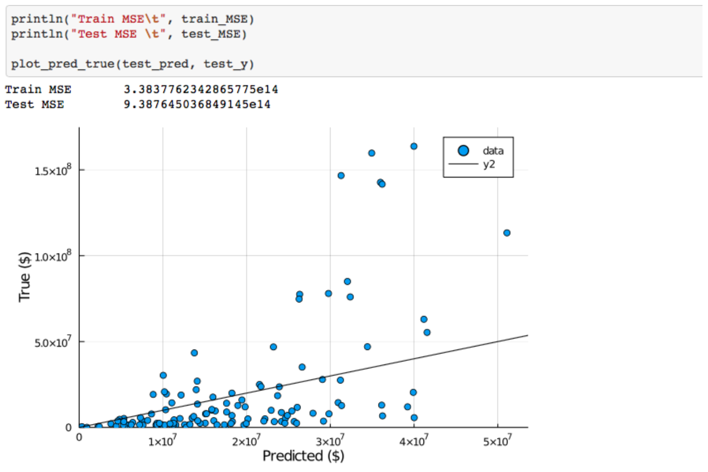


Figure 5: Running the perceptron algorithm when considering rebounds and blocks per game

It appears that regardless of what pair of features we choose to focus on, the resulting data will not be linearly separable, highlighting the fact that an entire host of features factor into a player's salary as opposed to just two statistics. To more closely explore which features most strongly impact a player's salary, we now turn to more complex models beyond the perceptron algorithm.

4 Linear Models

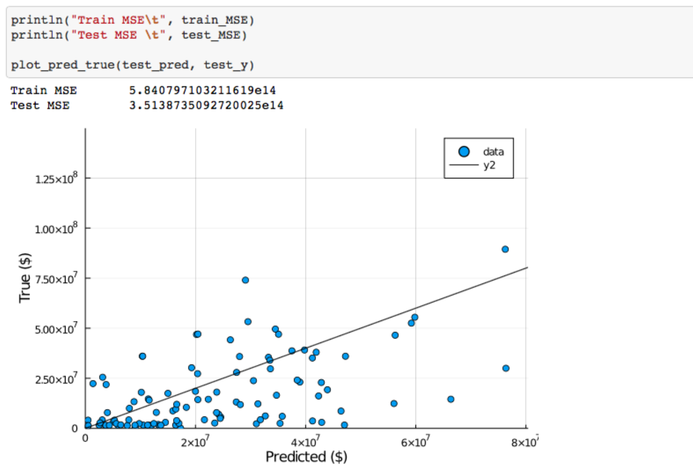
Initially, we developed a linear model that only accounted for points, rebounds, and assists per game (Figure 6). However, since this model resulted in an alarmingly high train mean squared error (MSE) of over 3.38×10^{14} and a test MSE of over 9.38×10^{14}, we decided to fit additional features to our linear model, specifically including blocks (BLK), minutes played (MP), effective field goal percentage (eFG%), and age in hopes of decreasing our MSE for both the training and testing data sets. Indeed, we found that the refined linear model led to a notable improvement in the cases of both the training and testing data (Figure 7). We now explore various loss functions and regularization methods in an attempt to improve the MSE values further.



```
println("Train MSE\t", train_MSE)
println("Test MSE \t", test_MSE)

plot_pred_true(test_pred, test_y)
```

```
Train MSE       3.3837762342865775e14
Test MSE        9.387645036849145e14
```

Figure 6: Initial linear model



```
println("Train MSE\t", train_MSE)
println("Test MSE \t", test_MSE)

plot_pred_true(test_pred, test_y)
```

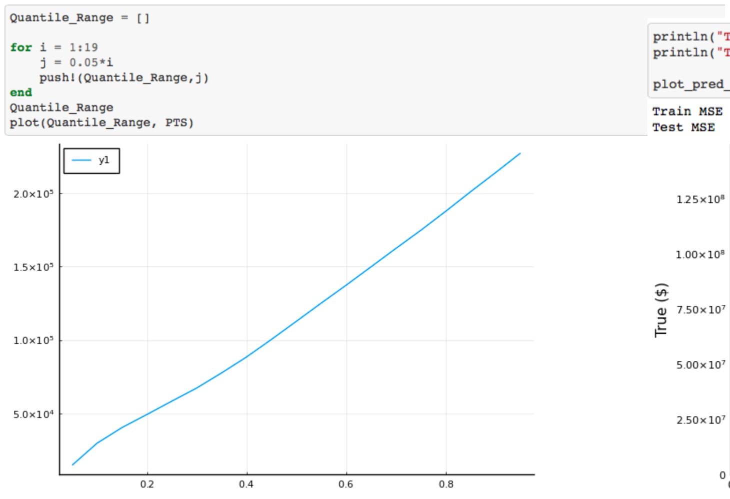
```
Train MSE       5.840797103211619e14
Test MSE        3.5138735092720025e14
```

Figure 7: Refined linear model

5 Initial Exploration of Loss Functions

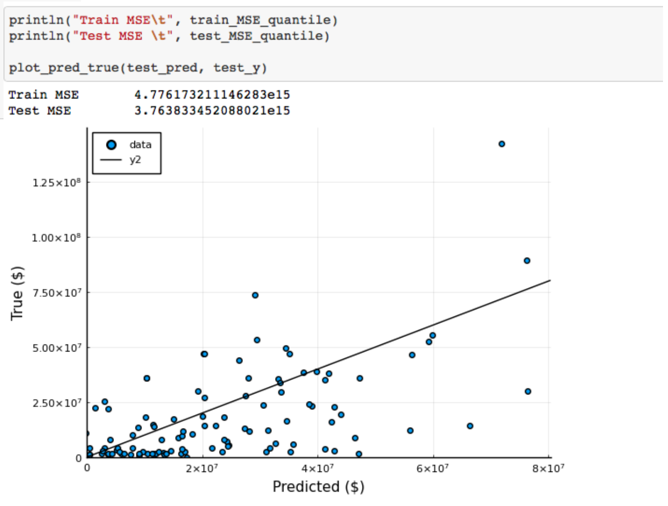
We began by exploring **quadratic loss with quadratic regularization**. Since the quadratic loss function is sensitive to outliers, we expect that using this loss function will largely be ineffective in the context of our dataset, given the myriad of situations in which outlier contracts are given. (For example, elite-performing players may still receive disproportionately low contracts in the case of an injury; and at the other extreme, seemingly poor-performing players may still receive disproportionately high contracts due to NBA front offices recognizing their potential to improve in the imminent future.) Nevertheless, we still confirm this is indeed the case by developing a model featuring quadratic loss and quadratic regularization.

As expected, using quadratic loss with quadratic regularization falls significantly short of fitting our data well, with the training MSE exceeding 6.81×10^{14} and the test MSE exceeding 4.58×10^{14}. To address this, we next turned to **quantile loss with l1 regularizer**. Using the proximal gradient function as well as a range between 5% to 95% for our quantiles, we plot the effect that various features have on our newly improved linear model. As seen below the trends seem to be very similar across all the features. As the quantile increases the points and Effective Field Goal Percentage (eFGPercentage) features have more of an effect on player's guaranteed salary on average. Since quantile loss are robust to outliers in the response measurements, it is expected that out MSEs for both the training and testing sets have negligible improvements at the very least. This seems to be the case when we fit the loss function in addition to a L1 regularizer to our data points using the standard least squares method.



```
Quantile_Range = []

for i = 1:19
    j = 0.05*i
    push!(Quantile_Range,j)
end
Quantile_Range
plot(Quantile_Range, PTS)
```

Figure 8: Quantile Plot for Points Featurepp



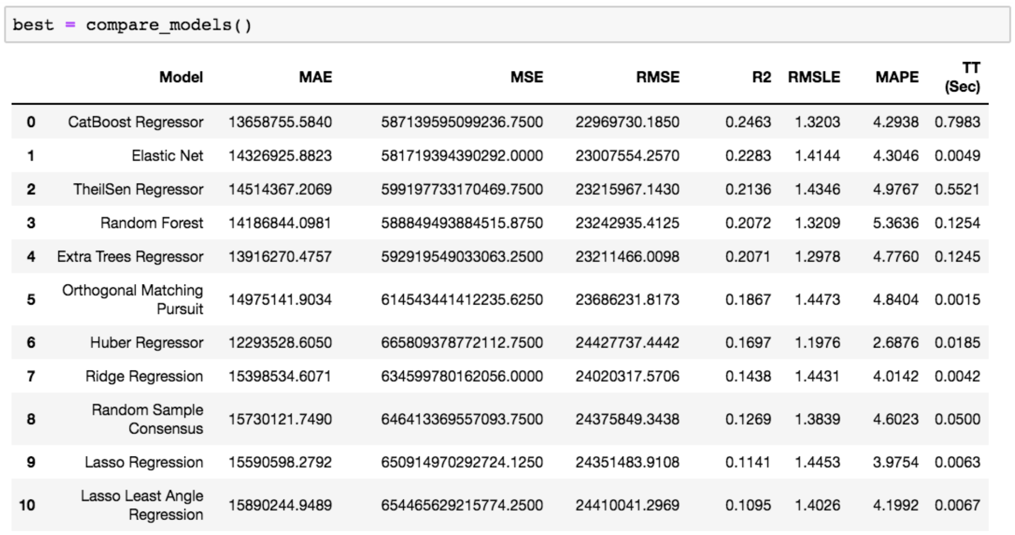
```
println("Train MSE\t", train_MSE_quantile)
println("Test MSE \t", test_MSE_quantile)

plot_pred_true(test_pred, test_y)
```

```
Train MSE       4.776173211146283e15
Test MSE        3.763833452088021e15
```

Figure 9: Quantile Loss MSE

6 Further Exploration of Loss Functions

After developing models around qudratic and quantile loss, we wanted to take a different approach to see which loss functions would perform the best by using the Python machine learning package `pycaret`. Since this package allows us to compare different models with mean-squared errors within few lines of codes, our goal was to find which model would perform the best, i.e., yield the smallest MSE:



```
best = compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
0	CatBoost Regressor	13658755.5840	587139595099236.7500	22969730.1850	0.2463	1.3203	4.2938	0.7983
1	Elastic Net	14326925.8823	581719394390292.0000	23007554.2570	0.2283	1.4144	4.3046	0.0049
2	TheilSen Regressor	14514367.2069	599197733170469.7500	23215967.1430	0.2136	1.4346	4.9767	0.5521
3	Random Forest	14186844.0981	588849493884515.8750	23242935.4125	0.2072	1.3209	5.3636	0.1254
4	Extra Trees Regressor	13916270.4757	592919549033063.2500	23211466.0098	0.2071	1.2978	4.7760	0.1245
5	Orthogonal Matching Pursuit	14975141.9034	614543441412235.6250	23686231.8173	0.1867	1.4473	4.8404	0.0015
6	Huber Regressor	12293528.6050	665809378772112.7500	24427737.4442	0.1697	1.1976	2.6876	0.0185
7	Ridge Regression	15398534.6071	634599780162056.0000	24020317.5706	0.1438	1.4431	4.0142	0.0042
8	Random Sample Consensus	15730121.7490	646413369557093.7500	24375849.3438	0.1269	1.3839	4.6023	0.0500
9	Lasso Regression	15590598.2792	650914970292724.1250	24351483.9108	0.1141	1.4453	3.9754	0.0063
10	Lasso Least Angle Regression	15890244.9489	654465629215774.2500	24410041.2969	0.1095	1.4026	4.1992	0.0067

Figure 10: Model Comparison Table

Above, Figure 10 shows the accuracy evaluation of each model from smallest to highest order. As we are looking at MSE, catboost regressor came in first in the list, having a MSE of approximately 1.37×10^7, a significantly lower value than the errors yielded by quantile and quadratic loss. We also looked at the significance of each feature in the model based on catboost loss, finding that the two features that had the most importance in the prediction model were number of free throws attempted per game (FTA) and number of points per game (PTS) (Figure 11).

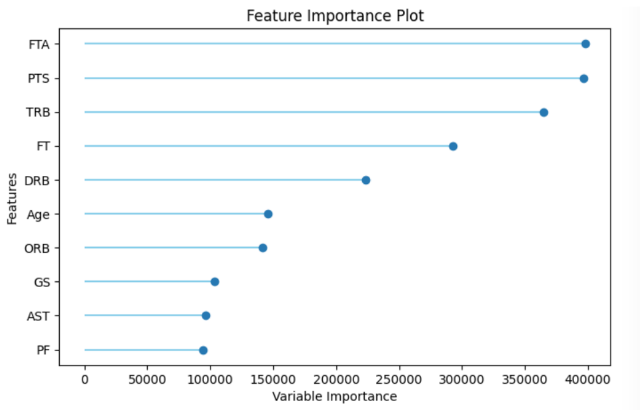


Figure 11: Feature Significance

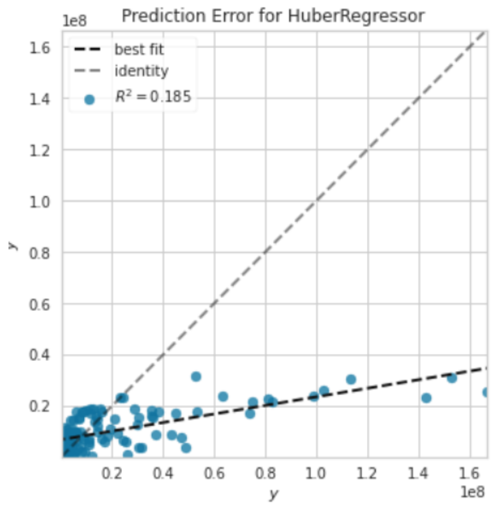


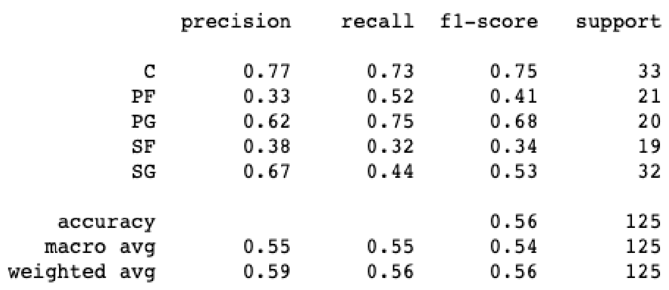
Figure 12: Huber vs. Catboost

Among the loss functions that we are most famliar with, huber loss ranked the highest, yielding an MSE of approximately 6.66×10^{14}. We directly compared huber with the catboost model in Figure 12.

Here, the dotted line with stronger stroke represents the catboost model, while the dotted line with lesser stroke represents huber loss. From this we can clearly see that catboost regession is a better fit to the data points than the huber loss.

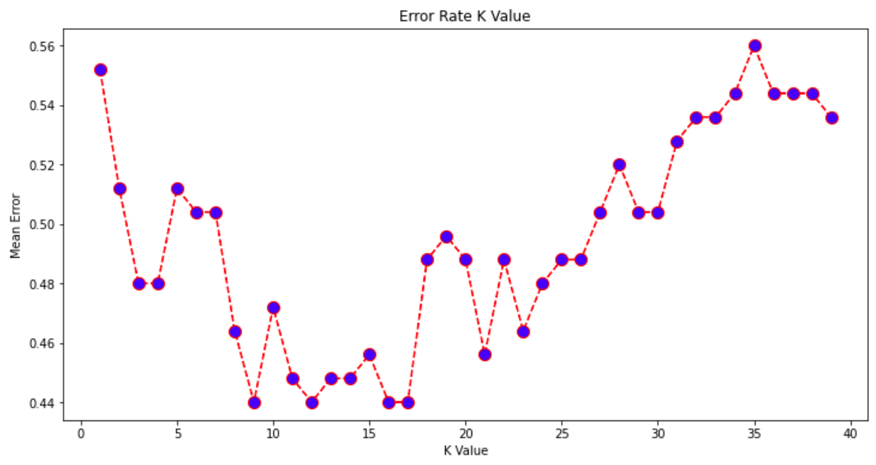
7 kNN

In order to account for positional discrepancies, we use kNN classification to separate each Player position by the points, rebounds, and assists, Blocks Per Game (BLK), Minutes per Game (MP), Effective Field Goal Percentage (eFGPercentage), and Age. In order to test the accuracy of our predictions with the kNN algorithm we scale the features using a Standard scalar and split the training and testing data with a 70% to 30% ratio. Using python's Scikit-Learn package, we can easily train the kNN algorithm and make predictions. Finally we make predictions on our test data and evaluate our algorithm. When evaluating the efficiency of a said algorithm commonly used involve a confusion matrix, precision, recall and f1 scores. Using the conduction matrix and classification methods from our Scikit-Learn package, we report the performance of kNN though the confusion matrix as detailed below. According to our confusion matrix, we find that our KNN algorithm was able to classify all the records in the test set with 56% accuracy. With the ambiguity surrounding NBA predictions based on our current data, these are promising results. Along with an impressive accuracy, we find that our KNN model yields similar accuracy metrics for many k values as highlighted in the error rates diagram for k values from 1 to 40 below. This shows that the kNN algorithm is a relatively effective model when accounting for the position discrepancies with the features used. kNN, however, is not expected to perform this well for all scenarios. Since our features consisted largely of real valued statistics, the kNN model improved substantially better than if one or more of our features consisted of categorial features as it would be hard to classify such data.



```
              precision    recall  f1-score   support

           C       0.77      0.73      0.75        33
          PF       0.33      0.52      0.41        21
          PG       0.62      0.75      0.68        20
          SF       0.38      0.32      0.34        19
          SG       0.67      0.44      0.53        32

    accuracy                           0.56       125
   macro avg       0.55      0.55      0.54       125
weighted avg       0.59      0.56      0.56       125
```

Figure 13: Condusion Matrix and Classification Report



The mean error is minimized when the K values are 9, 12, 16, or 17 .

Figure 14: Error rates with increasing k values

8 Conclusion

Overall, despite some promising results scattered throughout our models, we were ultimately unable to develop a model that—given contractual and statistical information for each player in the dataset— consistently and reliably predicts NBA salary. This likely pertains to the limitations of the dataset; for one, considering a set of 415 NBA players and their statistical performance in only one season (the 2017-18 NBA regular season) might have overly limited the scope of our models. Considering not only a larger set of players, but also their statistics across a number of different seasons (including playoff statistics) is certainly an intriguing idea to consider moving forward.

More importantly, however, the inconclusiveness of our models accentuates the fact that more features are necessary to predict NBA salary. Granted, our dataset does include a wide variety of both contractual information, per-game statistics, and personal player profiles; but there arguably remains a significant amount of room for feature expansion. Features such as injury history; previous contractual information; contract type (ex: fully guaranteed, partially guaranteed, non-guaranteed, ten-day, two-way, etc.); or year-to-year NBA salary cap numbers could all have plausibly improved our models' ability to definitively predict NBA salaries. In a sense, then, our findings here only serve to further highlight the surprising complexity behind NBA salary prediction—which is not too surprising given that hundreds of millions of dollars are at stake when NBA front offices give contracts to players.

In terms of weapons of math destruction (WMD), we concluded that our predictive models have some attributes characteristic of WMD. First, there are occasions where the NBA teams would pay higher guarantees to highly popular players on the team, although these players might not have performed that well statistically speaking. Therefore, a player's popularity is arguably an important feature necessary to take into account when making accurate salary predictions. However, any given player's popularity is clearly a subjective measure, which makes it difficult to represent in data for analytical purposes.

Aside from player popularity, players who are already being paid with high guarantees in previous years are likely to get relatively higher range of guarantees regardless of their performance on court. This again make our prediction model a WMD because the predictions can have feedback loops, in which highly paid players continue receiving progressively higher guarantees. However, there are still some factors that make our data not a WMD, such as the NBA draft lottery. Specifically, the NBA draft lottery allows the worst-performing teams every season to make the top draft selections, thus at least partially that the best-performing teams don't receive all the best players.

In terms of fairness, since all of our models are inconclusive, we can conclude that fairness is not so important to consider. Even if our models yielded more encouraging results, NBA teams likely would still not solely depend on our predictive models anyways to determine salaries, given the myriad of other practical factors necessary to decide each player's guarantees. But, hypothetically, if NBA team did used our predictive models to determine the player's guarantees, fairness would be important to consider in the interest of giving fair salaries to players, ensuring that the players are not discriminated by their race, national origins, injury history, etc.

References

[1] Renato Amorim Torres. Prediction of NBA games based on Machine Learning Methods. Project report. (2013), http://homepages.cae.wisc.edu/ ece539/fall13/project/AmorimTorres_rpt.pdf.

[2] Matthew Beckler, Hongfei Wang. NBA Oracle. Project report. (2008), http://www.mbeckler.org/coursework/2008-2009/10701_report.pdf.

[3] Scott Robinson. \K-Nearest Neighbors Algorithm in Python and Scikit-Learn." Stack Abuse, Stack Abuse, stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/.