Drew Adamski

Capstone 1 Final Report

An under discussed part of scouting, and one with great yields, is to use analytics to predict jumps in production from players that are already in the league. The central question of my project is as follows:

Given a player's statistics from a single season, can I predict whether that player will be an all-star the following year?

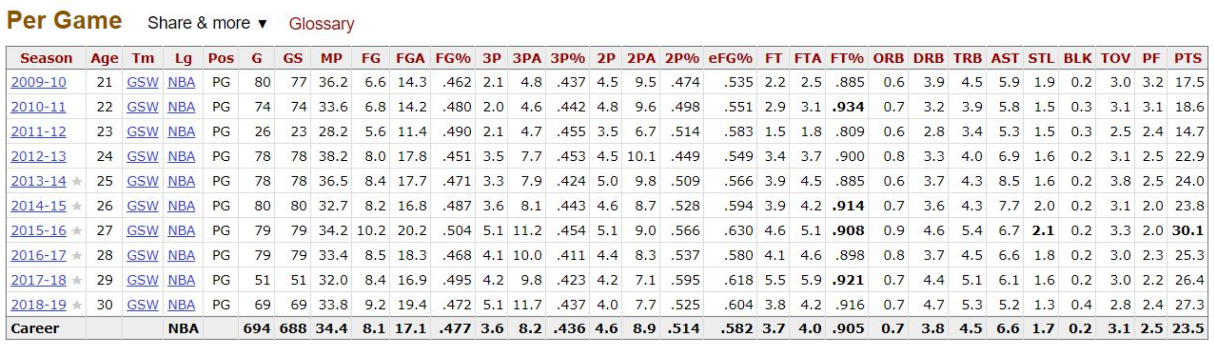
Answering this question greatly benefits general managers around the league who could implement new strategies of data analytics to identify players that may potentially break out the following season. Anything to give a team potential insight that other organizations do not have would be incredibly valuable to teams as they vie for the NBA Championship.

Data Wrangling

To answer this question I required accurate and complete season data in order to make my predictions. I was unable to find a preprocessed dataset online that fit my question, so I opted instead to create my own through web scraping.

I needed to find a basic list of players. I opted to use Basketball Reference's leaderboard for Hall of Fame Probability which is a statistic they come up with to measure exactly what it sounds like: a player's probability of making the Hall of Fame. I chose this subset of players to use because there would be a larger amount of all star appearances in this cohort, and I didn't necessarily want to build my model using lower level players who never made an all star game. This page includes all time leaders as well as active leaders, so I was able to get players from across the history of the league.

I scraped this list using the BeautifulSoup python package to pull player names and their personal statistics page info. I then iterated through this list of players and pulled season by season numbers for their entire career (Figure 1). I also included some other miscellaneous information like their height, position, team, and age. Each player's career data was then concatenated to a larger database containing every player's career info in the original list. Once every player's career data was in the final database, I began to poke around.

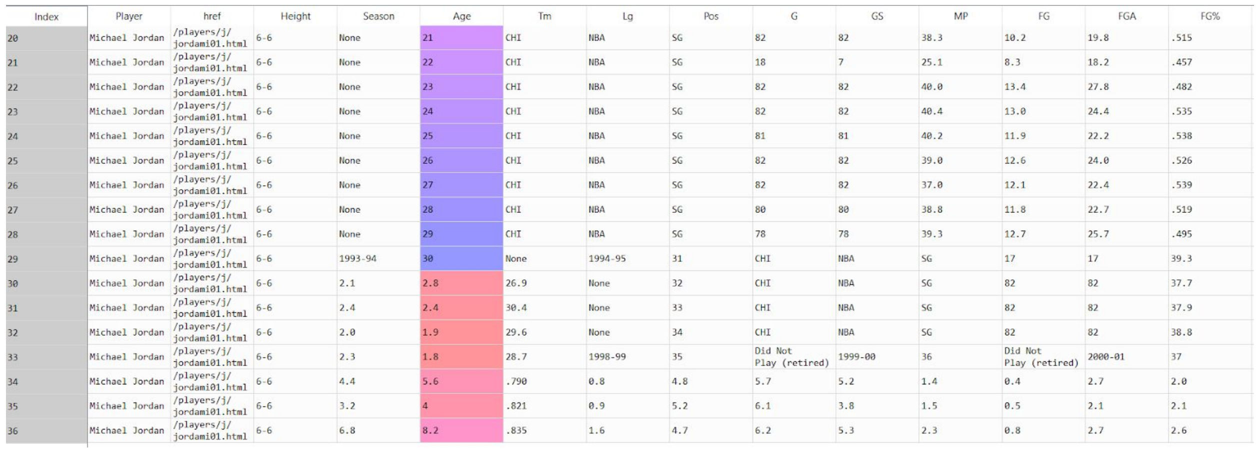


Per Game Share & more ▾ Glossary

Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
2009-10	21	GSW	NBA	PG	80	77	36.2	6.6	14.3	.462	2.1	4.8	.437	4.5	9.5	.474	.535	2.2	2.5	.885	0.6	3.9	4.5	5.9	1.9	0.2	3.0	3.2	17.5
2010-11	22	GSW	NBA	PG	74	74	33.6	6.8	14.2	.480	2.0	4.6	.442	4.8	9.6	.498	.551	2.9	3.1	**.934**	0.7	3.2	3.9	5.8	1.5	0.3	3.1	3.1	18.6
2011-12	23	GSW	NBA	PG	26	23	28.2	5.6	11.4	.490	2.1	4.7	.455	3.5	6.7	.514	.583	1.5	1.8	.809	0.6	2.8	3.4	5.3	1.5	0.3	2.5	2.4	14.7
2012-13	24	GSW	NBA	PG	78	78	38.2	8.0	17.8	.451	3.5	7.7	.453	4.5	10.1	.449	.549	3.4	3.7	.900	0.8	3.3	4.0	6.9	1.6	0.2	3.1	2.5	22.9
2013-14 ★	25	GSW	NBA	PG	78	78	36.5	8.4	17.7	.471	3.3	7.9	.424	5.0	9.8	.509	.566	3.9	4.5	.885	0.6	3.7	4.3	8.5	1.6	0.2	3.8	2.5	24.0
2014-15 ★	26	GSW	NBA	PG	80	80	32.7	8.2	16.8	.487	3.6	8.1	.443	4.6	8.7	.528	.594	3.9	4.2	**.914**	0.7	3.6	4.3	7.7	2.0	0.2	3.1	2.0	23.8
2015-16 ★	27	GSW	NBA	PG	79	79	34.2	10.2	20.2	.504	5.1	11.2	.454	5.1	9.0	.566	.630	4.6	5.1	**.908**	0.9	4.6	5.4	6.7	**2.1**	0.2	3.3	2.0	**30.1**
2016-17 ★	28	GSW	NBA	PG	79	79	33.4	8.5	18.3	.468	4.1	10.0	.411	4.4	8.3	.537	.580	4.1	4.6	.898	0.8	3.7	4.5	6.6	1.8	0.2	3.0	2.3	25.3
2017-18 ★	29	GSW	NBA	PG	51	51	32.0	8.4	16.9	.495	4.2	9.8	.423	4.2	7.1	.595	.618	5.5	5.9	**.921**	0.7	4.4	5.1	6.1	1.6	0.2	3.0	2.2	26.4
2018-19 ★	30	GSW	NBA	PG	69	69	33.8	9.2	19.4	.472	5.1	11.7	.437	4.0	7.7	.525	.604	3.8	4.2	.916	0.7	4.7	5.3	5.2	1.3	0.4	2.8	2.4	27.3
Career			**NBA**		**694**	**688**	**34.4**	**8.1**	**17.1**	**.477**	**3.6**	**8.2**	**.436**	**4.6**	**8.9**	**.514**	**.582**	**3.7**	**4.0**	**.905**	**0.7**	**3.8**	**4.5**	**6.6**	**1.7**	**0.2**	**3.1**	**2.5**	**23.5**

Figure 1: Example of the table from which I scraped the career stats of each player. In this case, Stephen Curry. Not included in this table are the number of pick up games ruined by people wearing Curry jerseys bombing 3's they have no business shooting.

One of the first issues I came across was that there were some seasons where a player would miss a full season due to injury, military service, or in misguided attempts to jumpstart their baseball career by batting .202 for the Birmingham Barons (Figure 2). These situations caused issues with my data scraping algorithm by shifting data a few columns over, so I had to go in and manually clean the data. Once all the data was placed in the proper columns, I needed to convert it to the correct data type. Now that my columns were filled in with data of the proper type, I could explore the actual values to make sure that they made sense.



Index	Player	href	Height	Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%
20	Michael Jordan	/players/j/jordami01.html	6-6	None	21	CHI	NBA	SG	82	82	38.3	10.2	19.8	.515
21	Michael Jordan	/players/j/jordami01.html	6-6	None	22	CHI	NBA	SG	18	7	25.1	8.3	18.2	.457
22	Michael Jordan	/players/j/jordami01.html	6-6	None	23	CHI	NBA	SG	82	82	40.0	13.4	27.8	.482
23	Michael Jordan	/players/j/jordami01.html	6-6	None	24	CHI	NBA	SG	82	82	40.4	13.0	24.4	.535
24	Michael Jordan	/players/j/jordami01.html	6-6	None	25	CHI	NBA	SG	81	81	40.2	11.9	22.2	.538
25	Michael Jordan	/players/j/jordami01.html	6-6	None	26	CHI	NBA	SG	82	82	39.0	12.6	24.0	.526
26	Michael Jordan	/players/j/jordami01.html	6-6	None	27	CHI	NBA	SG	82	82	37.0	12.1	22.4	.539
27	Michael Jordan	/players/j/jordami01.html	6-6	None	28	CHI	NBA	SG	80	80	38.8	11.8	22.7	.519
28	Michael Jordan	/players/j/jordami01.html	6-6	None	29	CHI	NBA	SG	78	78	39.3	12.7	25.7	.495
29	Michael Jordan	/players/j/jordami01.html	6-6	1993-94	30	None	1994-95	31	CHI	NBA	SG	17	17	39.3
30	Michael Jordan	/players/j/jordami01.html	6-6	2.1	2.8	26.9	None	32	CHI	NBA	SG	82	82	37.7
31	Michael Jordan	/players/j/jordami01.html	6-6	2.4	2.4	30.4	None	33	CHI	NBA	SG	82	82	37.9
32	Michael Jordan	/players/j/jordami01.html	6-6	2.0	1.9	29.6	None	34	CHI	NBA	SG	82	82	38.8
33	Michael Jordan	/players/j/jordami01.html	6-6	2.3	1.8	28.7	1998-99	35	Did Not Play (retired)	1999-00	36	Did Not Play (retired)	2000-01	37
34	Michael Jordan	/players/j/jordami01.html	6-6	4.4	5.6	.790	0.8	4.8	5.7	5.2	1.4	0.4	2.7	2.0
35	Michael Jordan	/players/j/jordami01.html	6-6	3.2	4	.821	0.9	5.2	6.1	3.8	1.5	0.5	2.1	2.1
36	Michael Jordan	/players/j/jordami01.html	6-6	6.8	8.2	.835	1.6	4.7	6.2	5.3	2.3	0.8	2.7	2.6

Figure 2: Example of a player's missing season affecting data scraping process. In one of sports more dumbfounding moments, following the Bulls third straight title and his third straight Finals MVP, Michael Jordan retired for the 1993-94 season to play baseball and then again from 1998-2001 before returning to play two more seasons for the Wizards. Those two Wizards seasons are underrated in hindsight. There aren't many players that could drag themselves up and down the court for an NBA game 4 days after their 40th birthday let alone drop 43 points.

I began exploring data by looking at the basic statistics to see if there were any extreme outliers that could signal additional errors in the scraping process. Through this I found missing decimal points, wrongly inputted data, and other minor glitches that were quickly fixed.

Something that I struggled with in my cleaned dataset was the issue of missing values. In the case of my data, the origin of missing data was that a lot of statistics were not counted in the earlier years of the NBA. Statkeepers began logging blocks and steal during the 1973-74 season, and it wasn't until 1979-80 that the three point line was officially instituted. What I ended up relying on to impute data was my domain knowledge. For instance, using the overall mean of three point attempts to impute data for a center in 1972 while a reasonable idea, does not make sense in the context of . My intuition was that there would be a difference between most statistics for backcourt players (PG, SG) and frontcourt players (SF, PF, C) and after running some z-tests on my data, my intuition was proven true. There was a significant difference for most statistics, so I imputed the mean for each subgroup.   My data was now in a place where I could begin to explore.

Exploratory Data Analysis

My first step was to take a look at the makeup of my target variable of whether a player was an all star the following year (labelled allstar_next). There were approximately twice as many non all stars as all stars which will impact how I build my samples to train and test my model. I will most likely perform oversampling as my pool of data is not incredibly large.

Next I wanted to take a look at the distribution of values for each statistic to understand what the probability densities looked like. I found that the distributions of the variables in my dataset took on a variety of forms (Figure 3). While many had approximately normal distributions, there were some surprising ones.

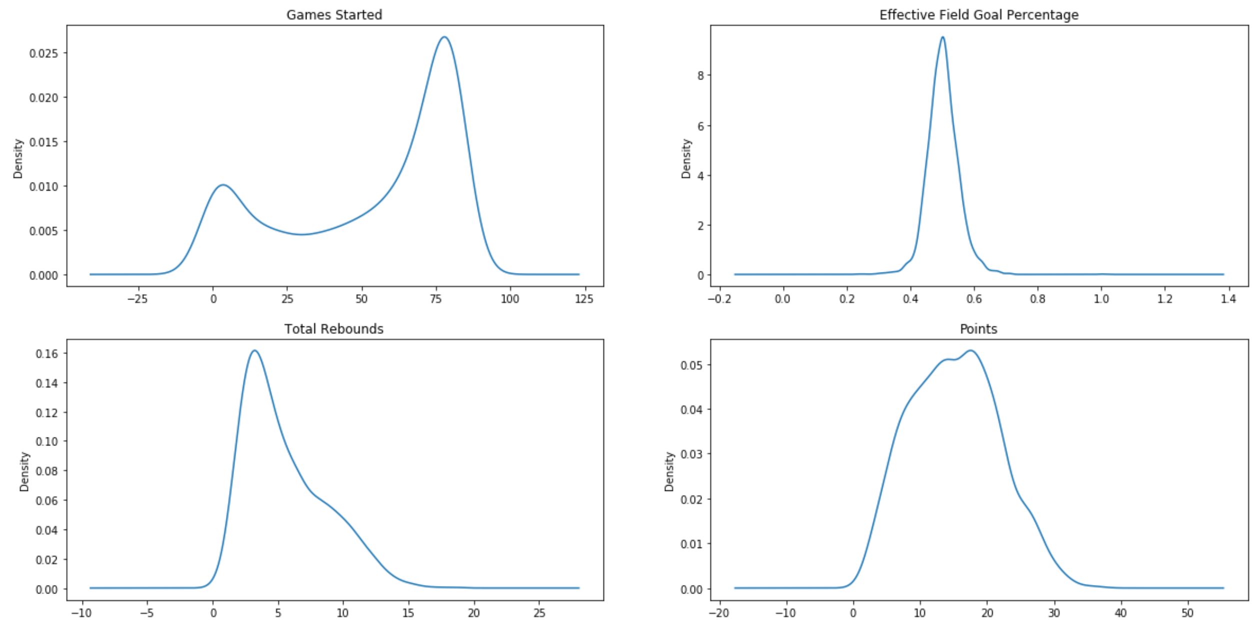


Figure 3: Examples of the sample distributions for the variables in my dataset. Games started has a somewhat binomial distribution since typically players are either starters or bench players. Effective Field Goal Percentage is a narrow, apparently normal distribution. Total Rebounds has a peak around x=4 and then drops quickly until around x=7 where interestingly, its slope levels out slightly. Points is a messy distribution that doesn't perfectly fit into a single description.

The whole purpose of this project is to make a prediction, so at this point it was time that I see what variables have a strong correlation with my target variable. I had created some scatter plots to see if there were any noticeable delineations of players that were allstars and those that were not (Figure 4).

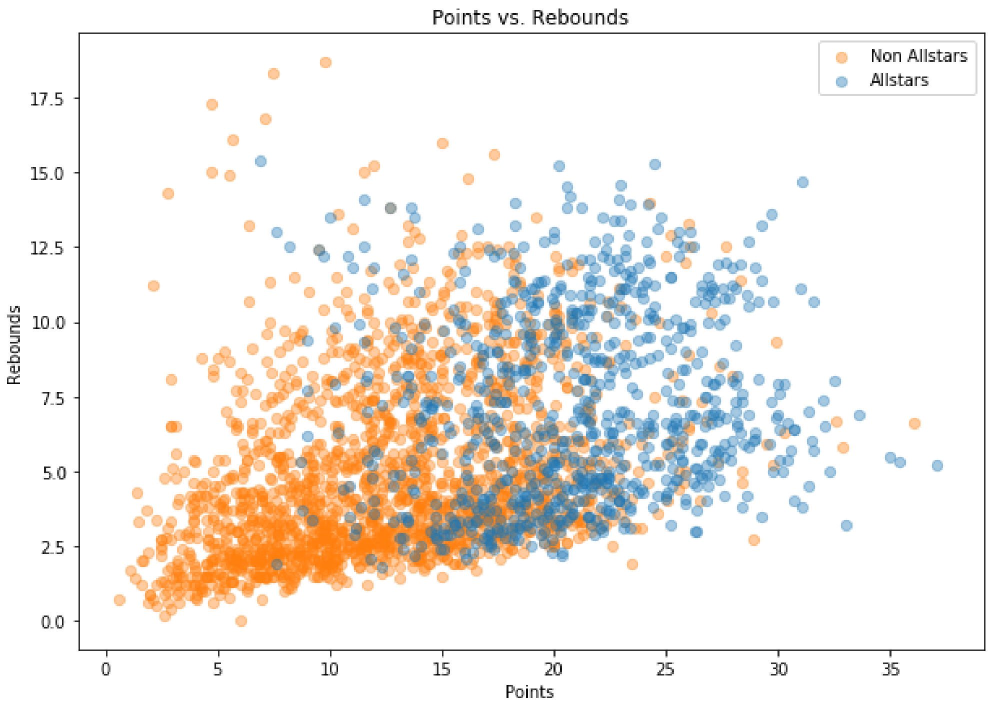


Figure 4: Scatter plot graphing points per game on the x-axis and total rebounds on the y-axis. Looking at this graph one does not see an explicit, well defined line of demarcation between allstars and non allstars, but there is potential to find something interesting.

I ran a correlation test on my data and extracted the allstar_next column exclusively to figure out the relationships. I found that all the columns associated with scoring (points, field goal attempts, free throws, etc.) had a strong, positive correlation with my target variable. It was in this moment that I realized there were definite problems with some of the columns of my data being highly related to each other in fundamental ways that I will need to be cognizant of when I build my model.

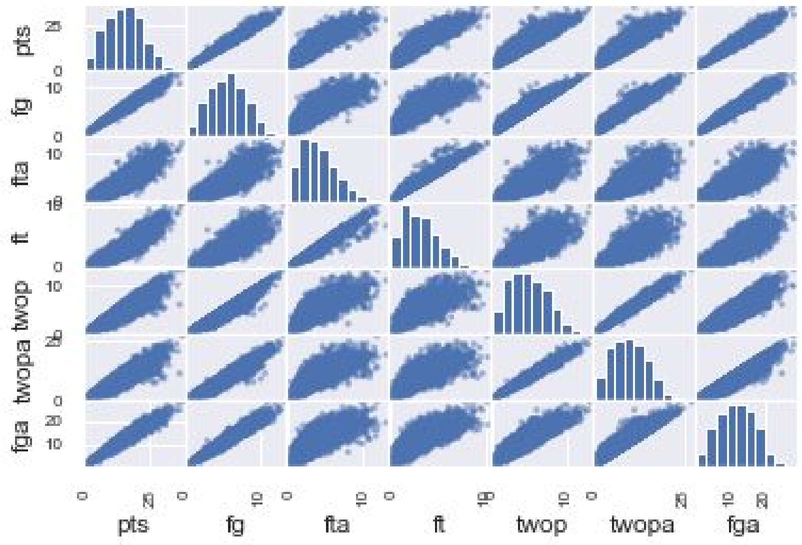


Figure 5: Correlation matrix between shooting statistics. These are all strongly correlated with each other.

Additionally, I took time to measure the general arc of a career. In order to do this, I looked at the proportion of all stars in at a given year in a player’s career (Figure 6). There is a bell like shape to it with steep incline in the early years, a peak lasting a few years, and then a steady decline until the end.

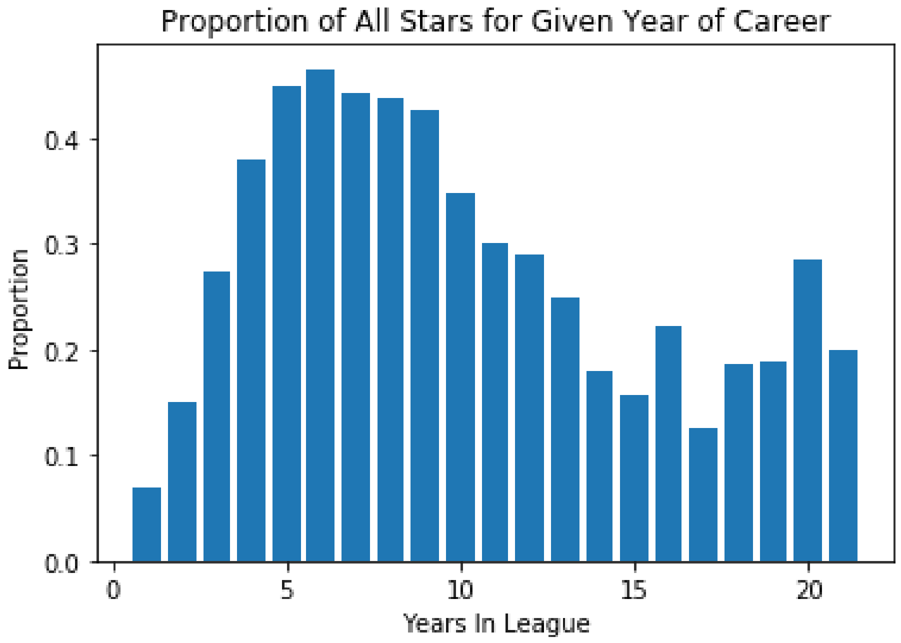


Figure 6: Bar graph showing proportion of all stars in a given year. This graph shows the general trend of a player’s career: The first few years are marked with improvement until around year 4-5 when they reach a peak level of play that lasts 4 to 5 years. Then there is a continued drop off until they eventually retire. The second increase in the later years is most likely caused by a bias in the data. Better players play for longer, so the pool of player data available at these ages comes from the greatest players to ever play basketball.

When I looked at the average value of some key statistics at different years, I saw a very similar pattern (Figure 7). There is a clear pattern between a player's performance in certain statistics, and it is one that I believe can be quantified.

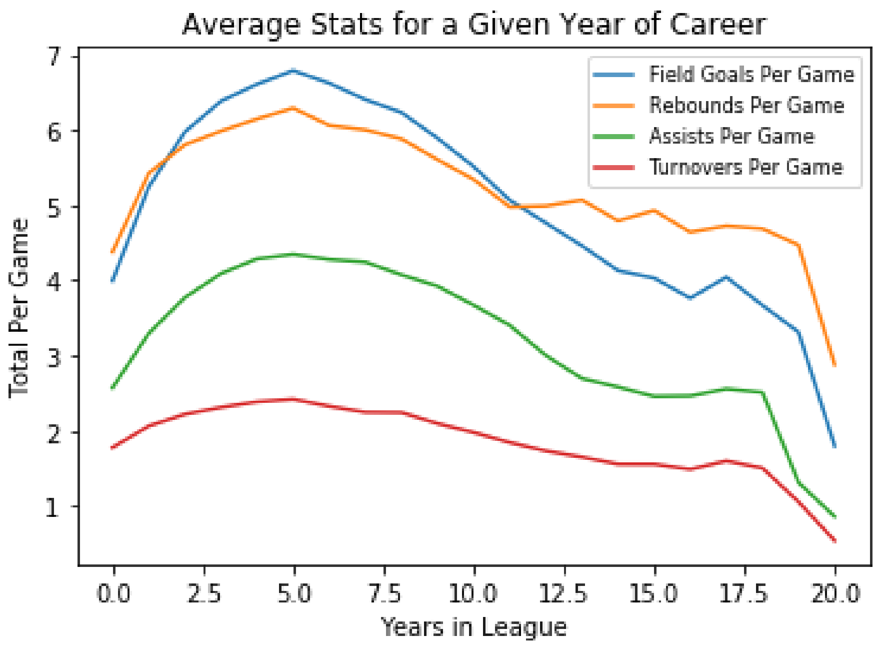


Figure 7: Graph charting field goals, rebounds, assists, and turnovers per game over time. These lines follow a strikingly similar pattern as the bar graph in Figure 6.

Feature Selection

There are two different methods of feature selection that I considered VarianceThreshold and SelectKBest.

VarianceThreshold removes all features with a variance equal to 0. Since I do not have any columns with the same values throughout, this will not help me reduce the dimensionality of the dataset.

SelectKBest selects a defined number of categories, k, that are most descriptive of the data. For classification problems, there are 3 types of tests that can be run to score the features: chi2, f_classif, and mutual_info_classif. chi2 uses a test to measure dependence between stochastic variables, so this function removes features that are most likely to be independent of class and therefore irrelevant to classification. However, it requires all positive values, and when I scale my data, some values are below 0. f_classif calculates the ANOVA F-value for the sample which is the ratio of two features' variance. Mutual_info_classif estimates mutual information for a discrete target variable. If MI between two random variables is 0, they are independent, and a high MI means higher dependency.

For my purposes, I used SelectKBest and use mutual_info_classif to determine the best features.

I split the data into the feature columns and the target column, allstar_next. I then separated each of these new dataframes into training datasets and test datasets. I used StandardizedScaler to preprocess the feature dataset. The target data is binary so no standardization is necessary. In order to prevent leakage in my model, I fit it to the training set only, and then use this fitted model to transform both the training and test sets. Figure 8 displays the best descriptors.

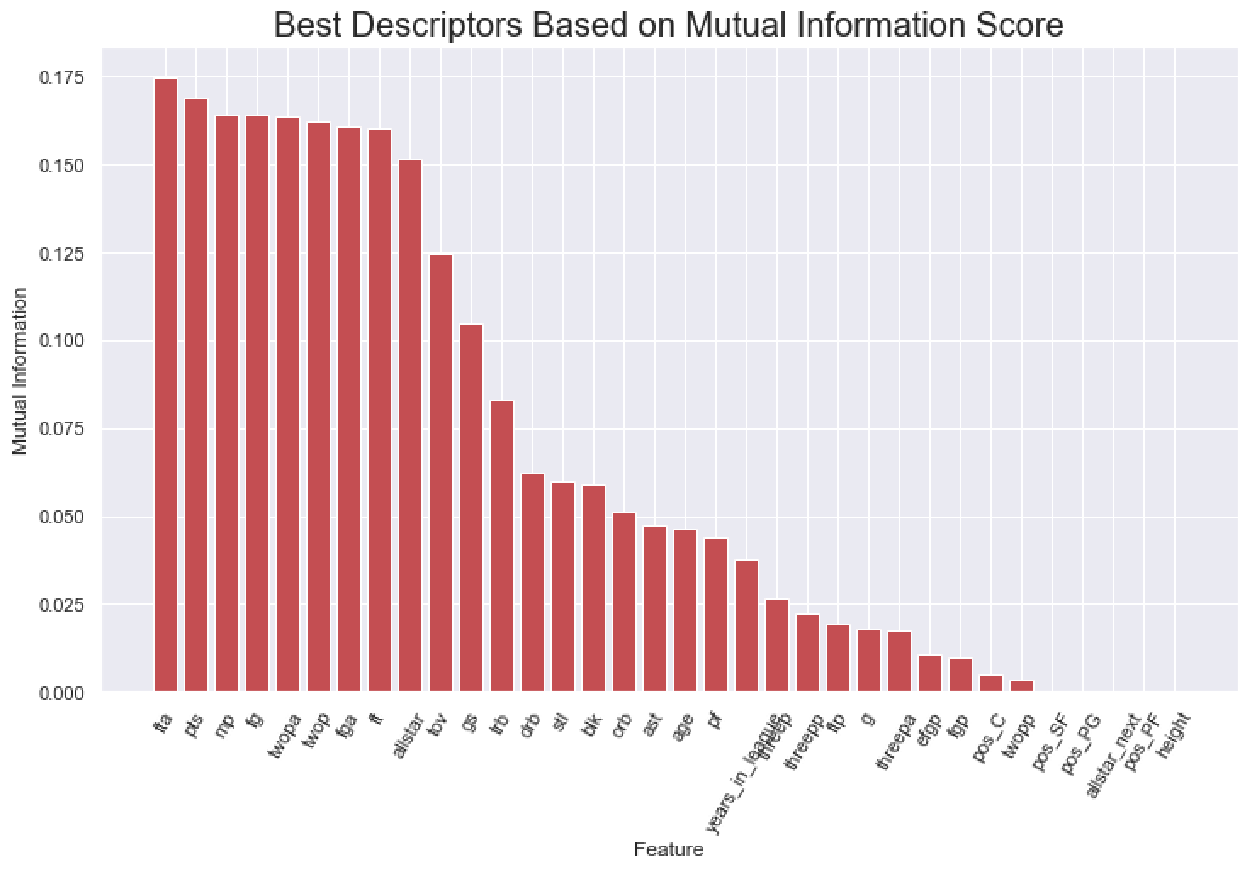


Figure 8: Graph of Best Descriptors Based on Mutual Information Score.

My 10 most descriptive features are: fta, ft, twop, fg, twopa, allstar, fga, mp, tov, and gs. However, I was not sure if reducing the number of features was the best option for me. In order to check this, I ran some tests on the ability of my model to correctly predict my data using different number of features. I performed Logistic Regression and compared F1 scores across different numbers of features to see if there was an ideal amount to use (Figure 9).

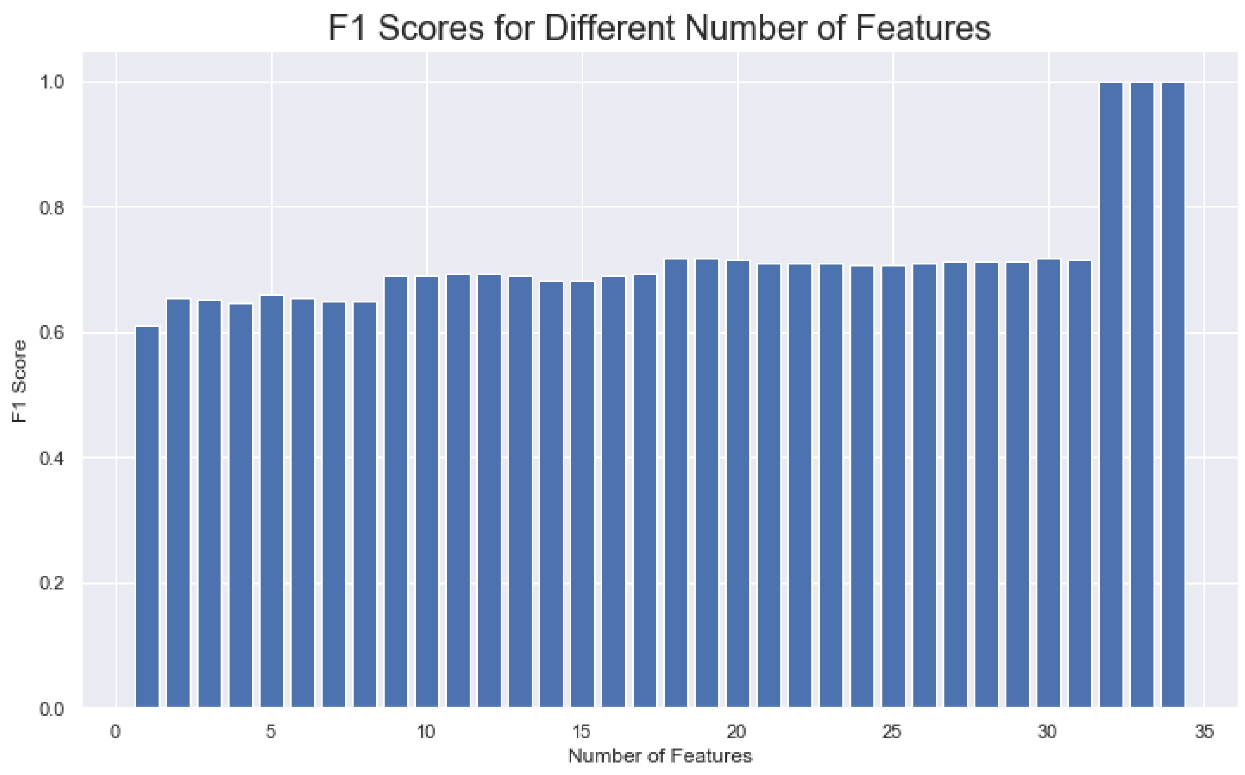


Figure 9: F1 Scores for Different Number of Features. Based on the results of this testing, I did not find it necessary to reduce the dimensionality of my dataset.

Since there is an initial increase in prediction power and then very small increases with more features until all features are then included, I am not going to reduce the dimensionality of the dataset further than I already have. Increased information will hopefully help me tune my model more effectively.

Model Selection

I tested 4 different machine learning algorithms:

- LogisticRegression
- RandomForestClassifier
- SVC
- KNeighborsClassifier

I tuned the hyperparameters each of these using GridSearchCV. Though I considered using RandomizedSearchCV, the parameter grids I created were not complicated to the point where Grid Search became time prohibitive.

Logistic Regression

There are a variety of parameters for logistic regression, but many of them did not apply to my question for a variety of reasons, and I chose those that did apply based on the scikit-learn documentation. For instance, while there are many potential solvers, many do not make sense for my project. sag and saga are for very large datasets. Given that I do not have an excess of columns, I want to use L2 penalty rather than L1 which could induce too much sparsity. Using newton-cg comes with many problems such as being computationally expensive and being attracted to Saddle Points where it can become confused about whether the input is a maximum or minimum. This leaves me with liblinear and lbfgs. I chose to use lbfgs because it is the default setting for LogisticRegression and served my purposes well.

I determined C and fit_intercept parameters to be most important for my model construction. I chose to use GridSearchCV over RandomizedSearchCV because the range of parameters and the size of my dataset is not so large as to cause problems. My scoring metric to determine the best model will be the F1 score because in my dataset, there are more seasons where the next year a player is not an allstar than there are seasons where a player is. Looking at accuracy alone would artificially inflate the models success rate.

Based off of the grid search results, my parameters for the model should be C = 10e8 and fit_intercept = False.

Random Forest Classifier

Next I used a Random Forest Classifier on my data. RandomForestClassifer has even more parameters than LogisticRegression, so I used RandomizedSearchCV for this testing. After looking through the documentation, I believe the following parameters are most important to do grid search on:

- **n_estimators** represents the number of trees in the forest. With more trees, the model better learns the data, but having too many leads to overfitting and slows down the training process considerably.
- **max_depth** is how many splits each tree will have. If it is too high, then the model overfits to the training data.
- **max_features** represents the number of features to consider when looking for the best split. At high values, it leads to overfitting.

To get an idea of what values to use for n_estimators and max_depth, I wanted to graph how the changes in their values affect the ability of my model accuracy. I used the ROC AUC to test, and based on my grid search, max_depth = 5 and n_estiamtors = 50

It would be valuable to visualize the relative importance of each feature in deciding the classification of each datapoint (Figure 10). The top three features make sense logically. If a player is an all star the previous year, there is a high likelihood they will repeat. Additionally, all stars are high scorers who play a lot of minutes for their teams.

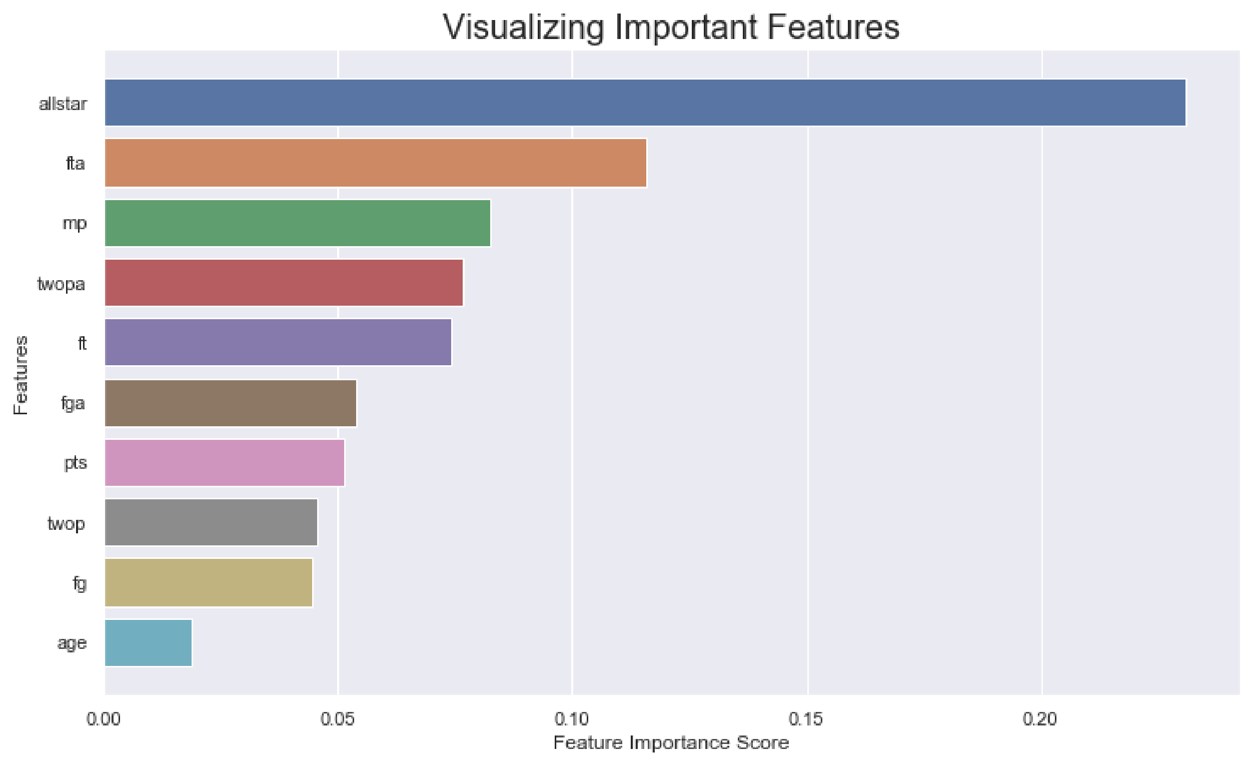


Figure 10: Feature Importance. Whether a player is an allstar the previous year has a huge bearing on whether they will be one the following year. As I saw in my analysis, this strong effect was keyed in on by all the models which partially explains the similarity of their predictive abilities.

Support Vector Machine

Support Vector Machines construct a hyperplane of set of hyperplanes in higher dimensional space in order to classify the data. For my model I looked at the value of the penalty term C and the kernel. There are a few different types of kernels available to use with Support Vector Machines, and I will be testing rbf and poly kernels to see which is best for my models. After performing GridSearchCV The best C value for SVM is 1 and the best kernel for me to use is rbf

K Nearest Neighbors Classifier

K Nearest Neighbors Classifier classifies points based the points surrounding it. Depending on the number of neighbors considered, a simple majority is taken and the point is then classified according to the winning category. If the number of neighbors considered is high, noise is less pronounced, but the boundaries are less defined. The opposite is true for a low number of neighbors, and overfitting becomes a problem.

I tested a variety of values for n_neighbors, p, and weights in order to fine tune my model. Through Grid Search Cross Validation, I selected n_neighbors = 30, p =1, and weights = 'distance'

Model Comparison

In order to understand which model performs best, I looked at Receiving Operating Characteristic and Precision-Recall curves. ROC curves measure how the True Positive Rate changes in respect to the False Positive Rate. The area under the curve is then calculated. Models with AUC closer to 1 are better than lower AUC. Precision-Recall curves show the change in Precision as Recall increases. AUC is used for this curve as well to compare models.

ROC curves are great for classification problems as are Precision-Recall curves, but ROC AUC can be artificially inflated in unbalanced datasets which can be mitigated by using a Precision-Recall Curve. I have created both since my data is a little unbalanced (Figure 11 and Figure 12).

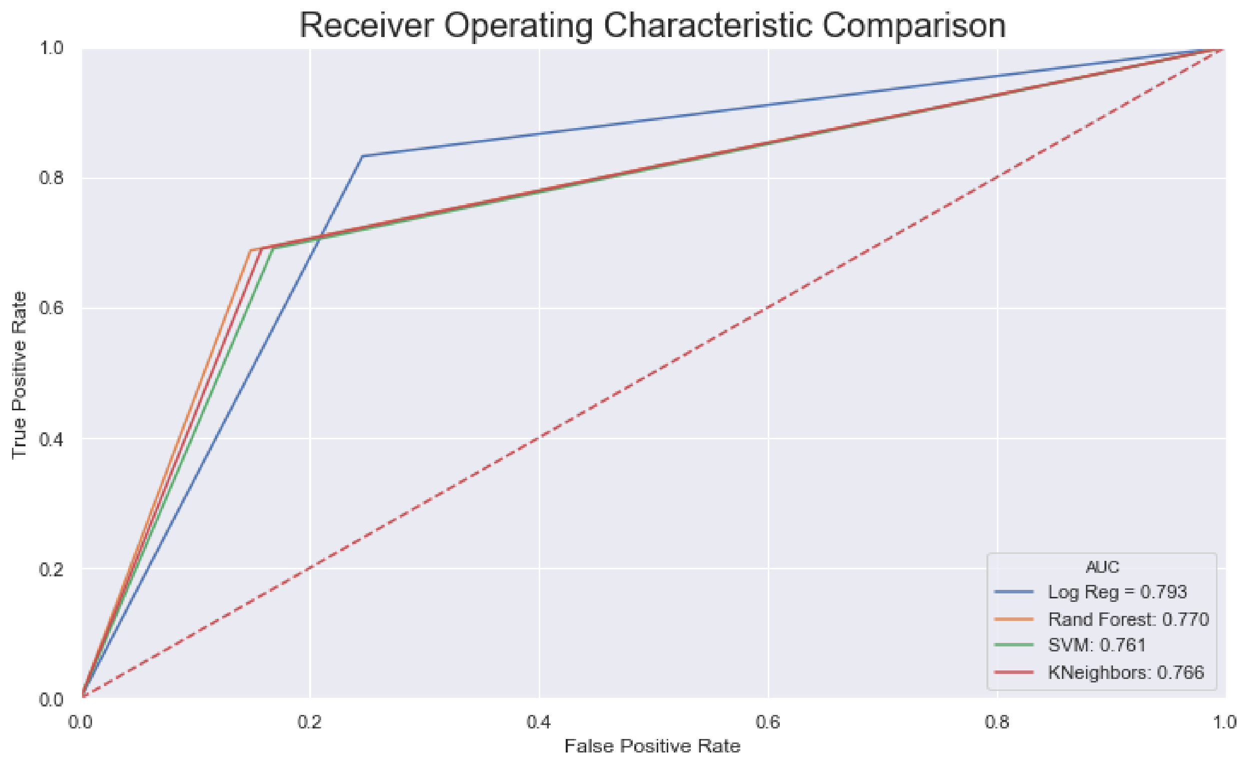


Figure 11: Receiver Operating Characteristic Comparison. The area under each curve is very similar, but Logistic Regression had the highest value of 0.793.

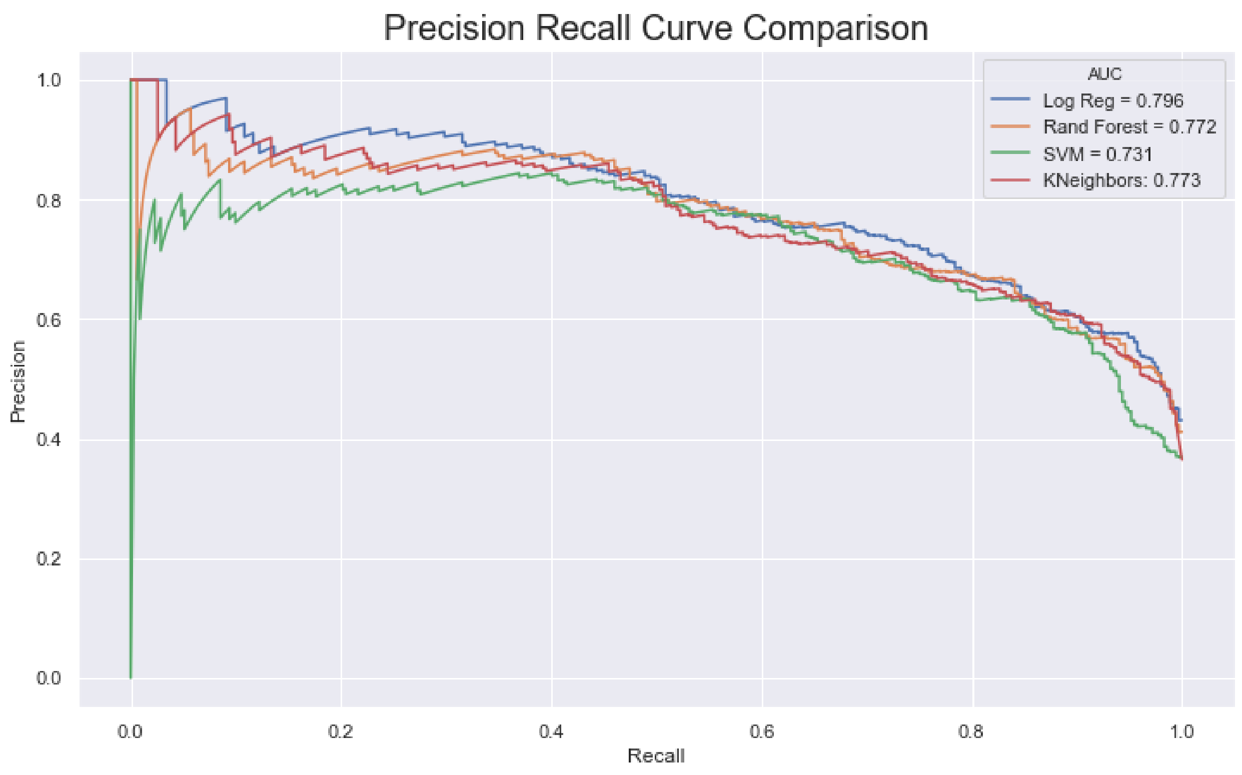


Figure 12: Precision Recall Curve Comparison. The curves had many similarities in their shape, but again Logistic Regression had the highest area under the curve value. With this metric, Support Vector Machines algorithm was worse than the previous measurement.

	Receiver Operating Characteristic Area Under Curve	**Precision-Recall Curve Area Under Curve**
Logistic Regression	0.793	0.796
Random Forest Classification	0.770	0.772
Support Vector Machine	0.761	0.731
K-Nearest Neighbors	0.766	0.773

Figure 12: Model Comparison of ROC-AUC and Precision-Recall AUC

Based on both metrics, Logistic Regression is the most effective model at predicting whether a player will be an all star the following year or not. The worst was Support Vector Machines. However, there was not a large difference in the overall prediction ability of any model, which means that they are most likely keying in on the same features to drive their classification.

Misclassifications

A huge question I have after modeling my data is what data points were misclassified. There may be a trend there that could help me identify why my model is not performing as well as I would hope. To do so, I am going to split my data and train each model. From there I will predict values and compare which seasons were misclassified across all three models.

There are 130 instances in my test set where all 3 models incorrectly predicted the value. This is a majority of the misclassified values, so if I can find a common thread running between these players, then I may be able to improve my model significantly. In order to check for trends in the misclassifications, I want to take a look at the difference in mean values between my misclassified samples and the entire dataset. As you can see below, the values of misclassified seasons are all right around the overall mean for the data. Intermediate values are difficult to parse out when compared to extremes.

An important case to focus on is new all stars. These are players who were not an all star one year but were the following. By being able to confidently identify this group of individuals, a team would be able to target players in free agency or trades and potentially pick them up for a discount.

Of the 94 first time all stars available in my test dataset, my model correctly predicted 41% of them to be all stars. This is a more difficult type of performance to predict, so 41% at this stage is pretty good. Perhaps with further refinement and a larger sample size to train the model, I would be able to increase that value.

Future Steps and Improvements

In order to further improve my model, I could increase the sample size to include more players. I only looked at the best players, so the general applicability of the model may be less than is shown here.

Additionally, I could add more advanced metrics such as Value Over Replacement Player (VORP) or Player Efficiency Rating (PER) to get more insight into what makes a player an all star.

With continued model refinement and expanded sample, I believe that this project could become an excellent predictor of player performance and prove to be a valuable asset when deciding what players to sign in free agency.