Predicting NBA Players' Positions Using Machine Learning Methods

Jerry Rong
Nikhil Sharma
Nikolas Thuy

Introduction

Basketball is a popular sport around the world with millions of fans dedicating their precious time watching professional athletes compete. International popularity also means monetary influx in various ways such as merchandises and sports gambling profit, with the latter being increasingly researched by computer scientists and statisticians alike. With the rise of machine learning and computing powers, more and more studies have been developed to predict the outcomes of NBA basketball games. The goal of this project is to develop a fundamental understanding of different machine learning methods and compare their accuracies when applied to official NBA statistics of individual players. More specifically, we wish to predict the position(s) that each individual NBA player plays from their statistical output. The result, or the target value, in our case will be one of five cases: Point Guard, Shooting Guard, Small Forward, Power Forward and Center. Thus, classification methods will be implemented to predict one of the five outcomes instead of regression, which is more suited to predict numerical values.

For machine learning methods to work optimally, it is imperative to acquire an accurate and clean dataset from the start. Luckily, baksetball-reference.com provides comprehensive data with player statistics starting from 1979-1980 all the way to the 2017-2018 season. The dataset we obtained (scraped from basketball-reference.com by Bruin Sports Analytics) has 50 columns and 19647 rows, with each row representing a player and each column representing a certain statistic. Thus, we only needed to adapt the relevant information: columns such as a player's age and the team that he played for were taken out of the dataset. We also chose to exclude statistics that are position-invariant, meaning that they do not define whether a player plays a certain position, including win percentage and number of games started. Next, results were again filtered to exclude players that had minimal roles on teams. We imposed a certain threshold in terms of minutes played to be greater than 15 minutes so that our analysis was done upon players that are impactful to the game. Missing values are also unavoidable in a large dataset. After thorough examination, we concluded that 3 point percentage as well as free throw percentage were the only columns with missing values. By intuition, we replace the missing values either with zero (in the case of three point percentage, suggesting that players that do not shoot 3's are likely not good shooters in general), or with the mean value for the specific season of players in the dataset (in the case of free throw percentage). Finally, the target values, represented in strings such as "PG" or "C" were given numerical values as labels. Note that players that designated with multiple positions were labeled with their primary position for simplicity, so if a player was marked as a PG/SG, he was considered a point guard for the model.

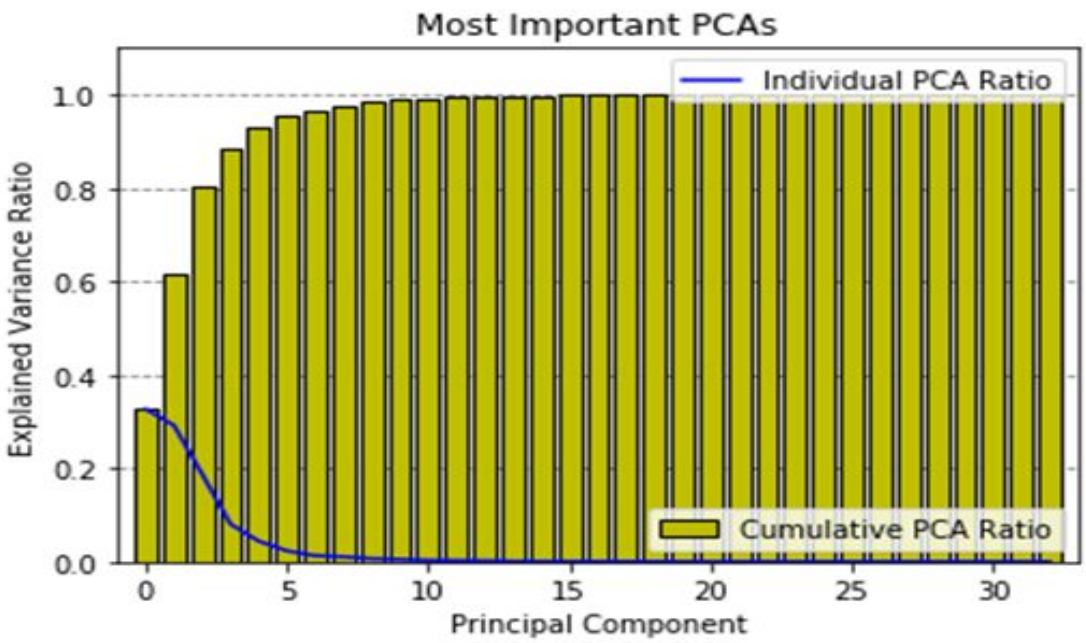
Here is a preview of the data:

player	pos	age	tm	g	gs	mp_per_game	fg	...	tovpercent	usgpercent	ows	dws	ws	ws_48	obpm	dbpm	vorp	i
Kareem Abdul-Jabbar*	C	32	LAL	82	NaN	38.3	10.2	...	15.7	24.1	9.5	5.3	14.8	0.227	4.0	2.7	6.8	1980
Tom Abernethy	PF	25	GSW	67	NaN	18.2	2.3	...	9.9	13.3	1.2	0.8	2.0	0.080	-1.5	-0.1	0.1	1980
Alvan Adams	C	25	PHO	75	NaN	28.9	6.2	...	18.2	21.9	3.1	3.9	7.0	0.155	1.6	2.8	3.5	1980
Tiny Archibald*	PG	31	BOS	80	80.0	35.8	4.8	...	19.7	17.0	5.9	2.9	8.9	0.148	1.1	-1.1	1.5	1980
Dennis Awtrey	C	31	CHI	26	NaN	21.5	1.0	...	24.8	7.9	0.1	0.5	0.6	0.053	-2.9	1.5	0.1	1980
Gus Bailey	SG	28	WSB	20	NaN	9.0	0.8	...	21.3	11.3	0.0	0.2	0.2	0.043	-3.7	1.3	0.0	1980
James Bailey	PF	22	SEA	67	NaN	10.8	1.8	...	20.0	21.4	-0.4	1.4	1.0	0.063	-4.5	0.9	-0.3	1980

Principal Component Analysis and K-Nearest Neighbors

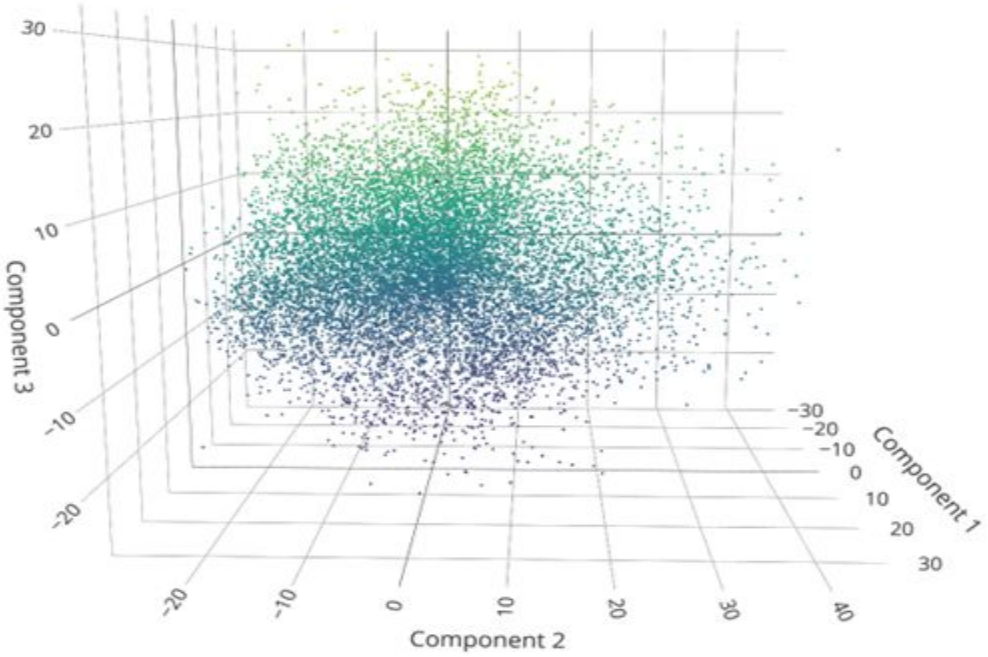
After data cleaning, the next step taken was to compress the remaining categories down into less columns, which was done to ultimately allow for visualization of a K-Nearest Neighbors model. There were thirty-three columns left in the dataset representing real statistics like field goal percentage and free throw percentage; to make three dimensional models, these were reduced to three dimensions using Principal Component Analysis (PCA). This was done by utilizing the PCA algorithm in the module sklearn, which outputted thirty-three principal components to match the number of columns started out with.

Using the function “pca.explained_variance_ratio_”, one could determine how much variance the first three components could explain:



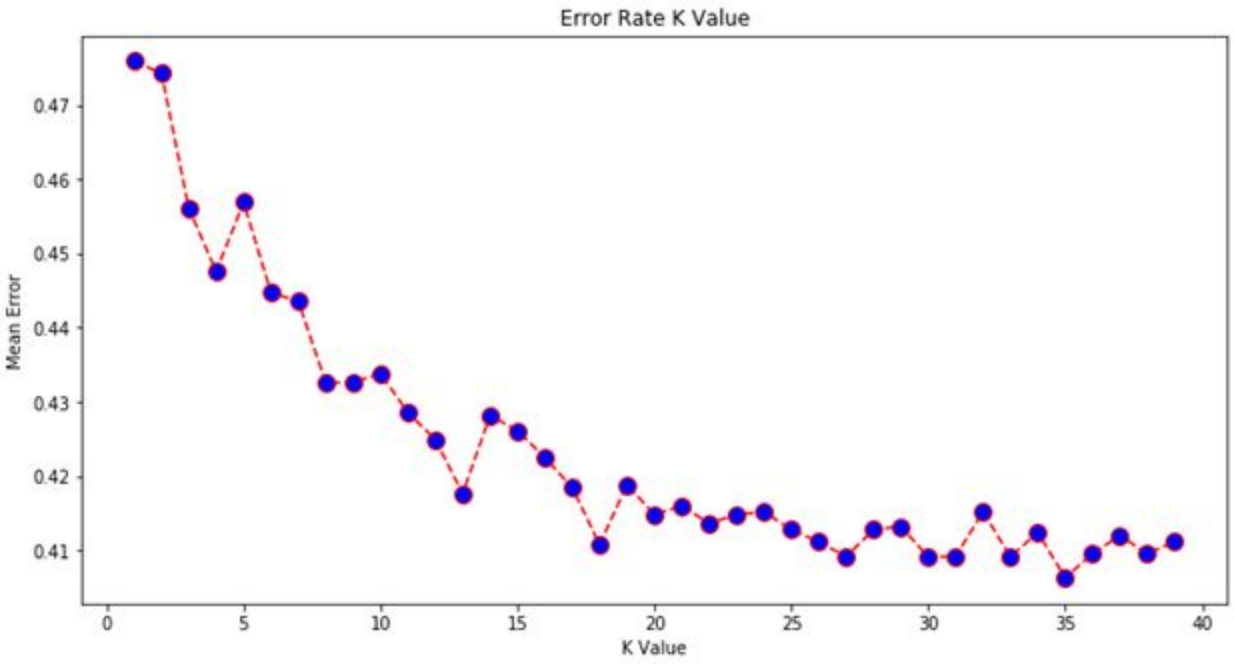
The chart expresses how since the PCA redistributed information among the columns in an unequal way, the first three columns of the PCA taken together explain about eighty percent of the variance in the dataset. So, it seemed reasonable to feed to a machine learning algorithm only these three most principal components since more than a majority of the information provided by the original, cleaned dataset would be retained.

Following this, each player was placed on a plot.ly scatter plot based on their three values for the three principal components employed. This outputted what looked like a blob of data points because no machine learning models were used to separate the data yet:



The actual plot.ly graph can be seen here: https://plot.ly/~nthuy/24.embed , where one can hover over a point to check its corresponding player's name and position.

The machine learning model that was using these components was the K-Nearest Neighbors model. After testing out various values of K to use, it seemed that after around K=20, the accuracy of the model stayed relatively stagnant at around 60% accurate. This can be seen in the following chart:



So, the following results used K=26, meaning that the nearest twenty-six data points with uniform weighted values were used in classifying a new datapoint. Using the function "confusion_matrix" from sklearn allowed for one to see how the KNN algorithm classified players incorrectly:

	Predicted PG	**Predicted SG**	**Predicted SF**	**Predicted PF**	**Predicted C**
Actual PG	454	61	3	0	0
Actual SG	88	307	144	16	1
Actual SF	9	108	246	76	33
Actual PF	2	19	114	202	169
Actual C	3	7	45	115	249

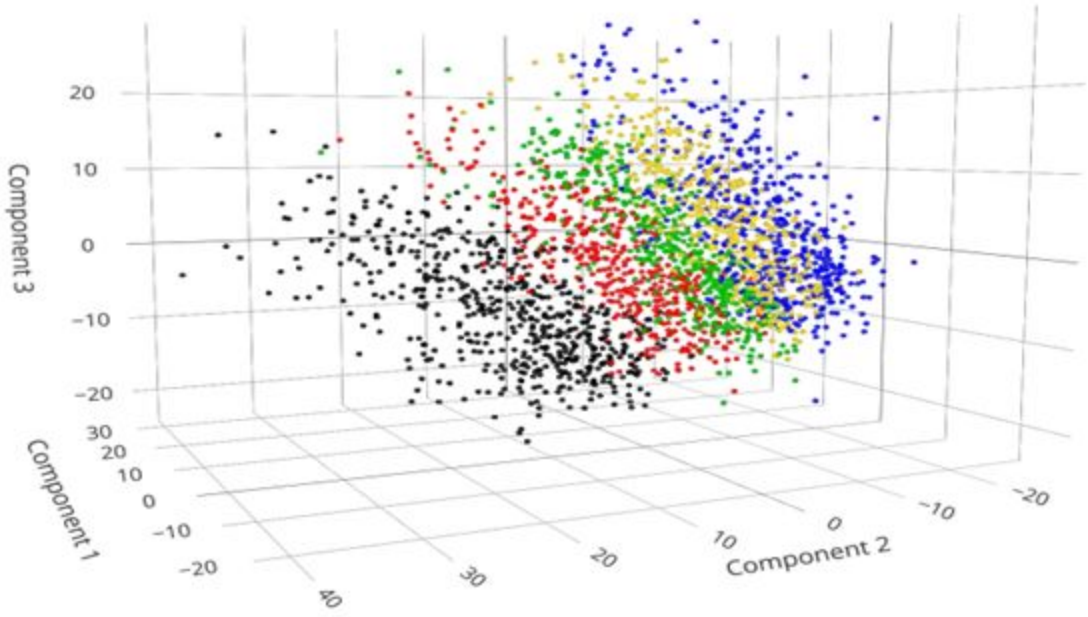
For example, the machine correctly classified 249 centers as centers, but mistakenly classified 115 centers as power forwards and three centers as point guards. This can be better represented in percentage form, which was provided using the function “classification_report”, also from sklearn.

	Precision	Recall	F1-score	Support
PG	0.82	0.88	0.85	518
SG	0.61	0.55	0.58	556
SF	0.45	0.52	0.48	472
PF	0.49	0.40	0.44	506
C	0.55	0.59	0.57	419
Micro Avg.	0.59	0.59	0.59	2471
Macro Avg.	0.58	0.59	0.58	2471
Weighted Avg.	0.59	0.59	0.59	2471

The KNN model predicted point guards most accurately at a rate of around 85%. It was found out that the most predictive column in the cleaned dataset included defensive rebound percentage and assist percentage (which will be shown later in the report when feature importance is discussed). It seems that this machine learning model came to a similar conclusion since point guards usually have the least amount of rebounds due to their height and also the most amount of assists due to their ball dominance.

The actual 3D scatter plot can be seen in the following link: https://plot.ly/~nthuy/18.embed , and a snapshot is given below. The black dots represent a player predicted to be a point guard, the red dots represent a player predicted to be a shooting guard, the green dots represent a player predicted to be a small forward, the yellow dots represent a player predicted to be a power forward, and the blue dots represent a player predicted to be a center. If one hovers over

a point in the actual plot on plot.ly, then one can see the actual position of that player in order to compare that information with what was predicted. Due to how K-Nearest Neighbors classifies data based on the values of the nearest K neighbors from the training set and how five clusters of colors seems to be present in the below picture, it seems that the PCA itself reasonably separated players into positions based on principal components, however, having only a 58% accuracy meant that better results could be obtained.



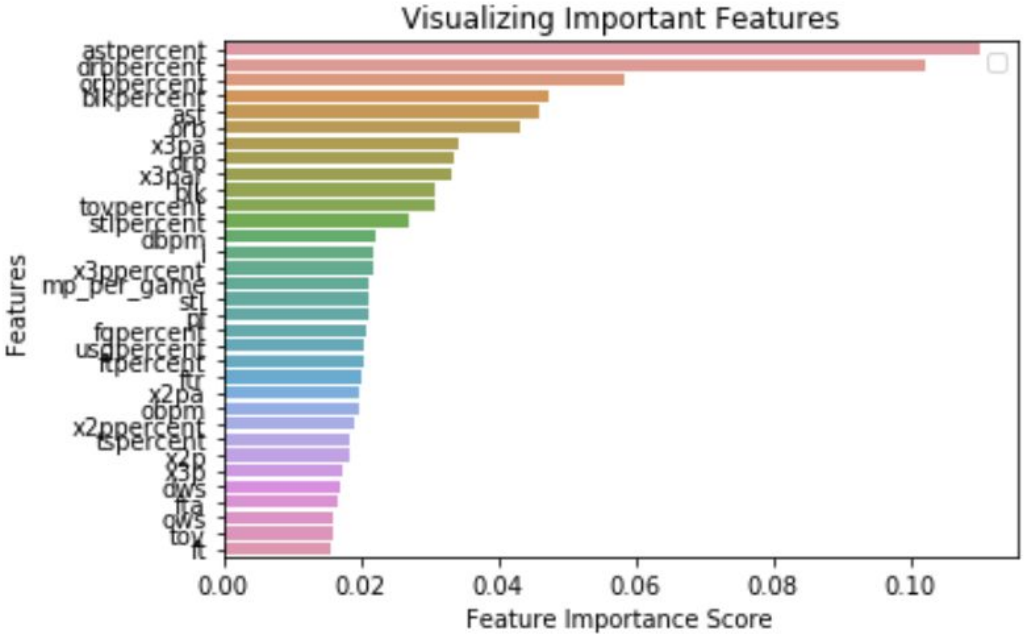
Decision Tree and Random Forest Classifier

Since our KNN with dimensionality reduction approach did not churn out a model with the predictive power we wanted it to, we decided to go a different route. First, we resorted to building a decision tree. A decision tree is a machine learning model which features nodes and branches. Each internal node represents a feature while each branch represents a decision. A collection of nodes and branches comes together to form a tree which the algorithm partitions based on the given data. This can make a decision tree very easy to visualize, as one can easily go down the branches and evaluate features to see where prospective data points would land in their model. Unfortunately, decision trees are sensitive to small variations in data and can get very complicated, as we saw when we fit a decision tree to our data:

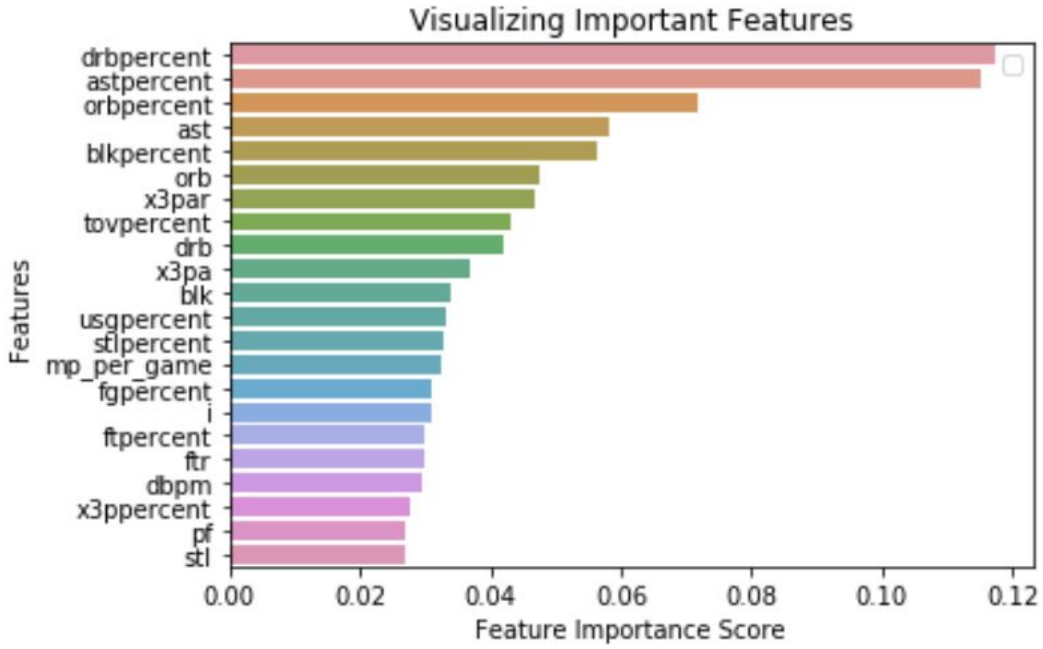
This tree is practically unreadable; even when zoomed in, it does not give any insight as to how different positions are classified in different ways. Also, its accuracy rate was only about 60.50%, so we did not want to continue with this model. However, an important thing we did

gather from this model was that the decision tree constructed was a complex one that almost looked like a series of connected, smaller decision trees. In seeing this, we decided to use an ensemble model to tackle the problem.

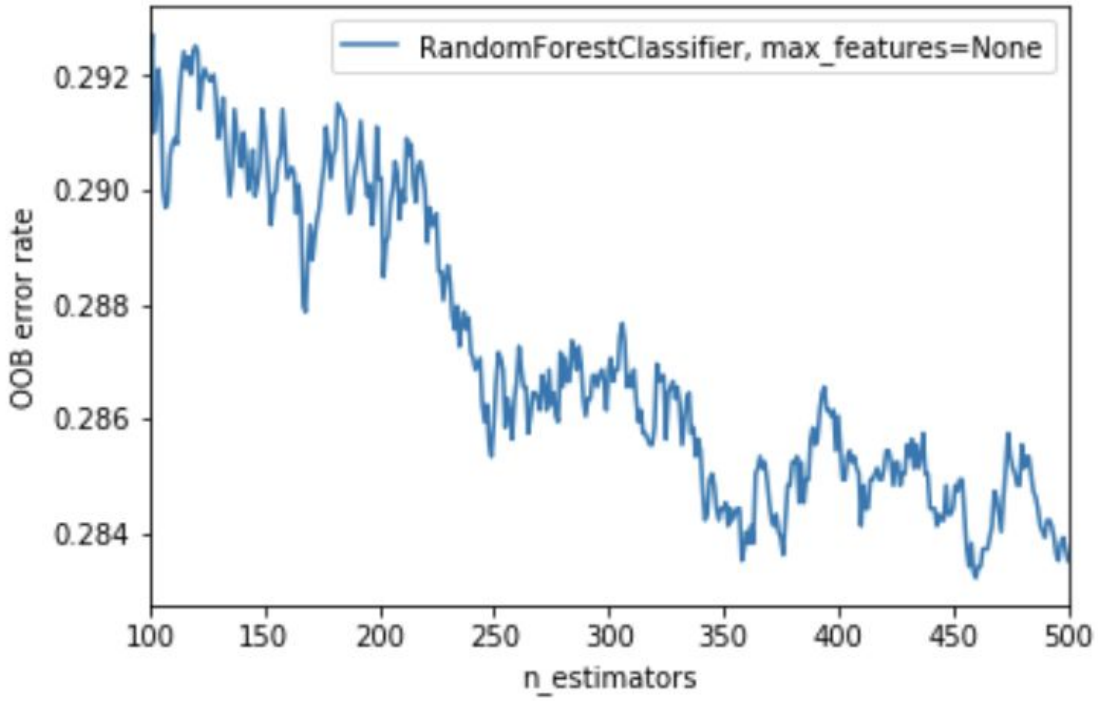
We chose to fit the data with a random forest model. A random forest is an ensemble machine learning model that partitions the training set into many samples via bootstrapping (sampling within the training set with replacement), and constructs a decision tree for each sample. Then, for each data point in the testing set, the random forest evaluates that data point on every classification tree built, and classifies based on a majority vote. A great thing about random forest is that it can tease out feature importance of each column in your data, so we could see which specific features were most important in teasing out position. On our initial try of the model, we got this variable importance plot:



To attempt to reduce some of the unnecessary variables, we chose an arbitrary cutoff of 0.02, meaning that the variable was taken out if its feature importance score was less than 0.02.



We ended up reducing the data down to 22 columns from 33. Then, we tuned the model further by deciding how many trees were appropriate in construction of the random forest.



As we can see here, the error, as expected, generally goes down as the number of trees increases. However, looking on the x-axis, this difference in error rate is quite marginal, and the

algorithm gets far slower and thus more computationally expensive as we add more trees. For reference, a 100-tree model had an accuracy of 72.64% with a runtime of 2.109 seconds; a 300-tree model had an accuracy of 73.17% with a runtime of 6.039 seconds; and finally, a 600 tree model had an accuracy of 72.52% with a runtime of 12.33 seconds. So, in this iteration, the simple 100-tree model actually had a greater accuracy than then complex 600-tree model. While this will not always be the case, it is clear that the computational simplicity vs. accuracy trade off favors computational simplicity, thus, we stuck with the 100-tree model.

Analysis/Conclusion

Here is a simple confusion matrix for our classifier:

	Predicted PG	Predicted SG	Predicted SF	Predicted PF	Predicted C
Actual PG	350	93	0	8	0
Actual SG	100	297	0	69	4
Actual SF	0	0	471	4	57
Actual PF	13	60	8	325	93
Actual C	0	4	62	93	360

As we see, our model makes some pretty intuitive errors. The next most predicted position for actual point guards was shooting guards (and vice versa), which makes sense because both assume the guard role and thus often perform similar roles on the team, including fronting the load on playmaking on the offensive side of the ball. The next most predicted position for power forwards was centers (and vice versa), which also makes sense since both play the big man role and have similar roles including scoring in the paint and anchoring the defense from the same area.

Our random forest model teased out Defensive Rebound Percentage (percentage of defensive rebounds grabbed by a player when he was on the floor) and Assist Percentage (percentage of assists dished by a player when he was on the floor) as the two most important features in classifying a player's position. Here are some summary stats for those metrics per position:

pos	Mean Defensive Rebound %	Standard Deviation
C	20.635795	4.418387
PF	18.691587	4.439482
SF	12.969980	3.356313
SG	9.632455	2.868742
PG	8.801736	2.566520

pos	Mean Assist %	Standard Deviation
PG	27.867631	8.154549
SG	15.384121	6.329068
SF	11.687343	5.228567
PF	9.159649	4.632811
C	7.822700	4.298629

As we see, defensive rebound percentage is a key differentiating stat for big men, as centers and power forwards on average have higher defensive rebounding percentages than the other three positions. On the other hand, assist percentage is a key differentiating stat for guards, especially point guards, as guards on average have higher assist percentages than the other three positions. This helps tell us why our model had some trouble in differentiating guards and big men, while predicting small forwards very well; the two most important features in the construction of the random forest were similarly distributed for point guards and shooting guards and power forwards and centers, indicating some room for error. On the other hand, the model might have predicted small forwards relatively well since their distributions stand out as apart from the pack for both key statistics, meaning the model would have less trouble differentiating them from other positions.

Further Studies

This project clearly has room for improvement, since our best model stood at an accuracy of 72.64%. One possible further study could be to partition the positions into smaller groups; perhaps group guards, big men, and wings, and see if the model works better classifying those three versus the five generally accepted basketball positions. Another interesting further study could be to use unsupervised methods to classify general playing styles, thus creating our own positions (see here) based on statistical groupings of playing styles. From there, we could apply the random forest model we built and see how predicted old positions and formulated new positions compare and contrast. Overall, the model we have built is a good basis for initial studies on positional analysis in the NBA through machine learning.

Source Code

Data:
https://github.com/bruinsportsanalytics/Resource-Folder/blob/master/Data/Basketball/regular_season_players.csv

Cleaning the data: https://codeshare.io/2KXJRz

PCA: https://codeshare.io/arwz87

PCA Plot: https://codeshare.io/5O0KW0

KNN: https://codeshare.io/2jYZzD

KNN Plot: https://codeshare.io/50MAVm

Random Forest: https://codeshare.io/5NQDMV