NBA All-Star Predictor

Anthony Song, Grant Westfall, Justin Cho, Zachary Joseph

March 2023

1 Introduction

Every year, a new group of rookie players joins the NBA, bringing with them different perspectives and skill sets. Some of these rookies go on to become all-stars and members of the Hall of Fame, but some find it difficult to have a big impact. We are always looking for new and innovative ways to predict which rookies will become the next big stars in the league. One promising approach is to analyze player data to identify key indicators of success.

The study of predicting NBA career success from rookie seasons is of significant importance for several reasons. Firstly, it can be greatly helpful to general managers and their front offices in the league. General managers often have to make difficult decisions in the draft and free agency based on the perceived value of young players on their roster. They are, in effect, attempting to forecast the future of their current players in order to determine which players to obtain and release. A rigorous statistical approach can help them to make more informed forecasts and decisions. Secondly, these predictions are important for fans and analysts who are looking to identify the next generation of NBA stars. The sport attracts a lot of time and money from fans, who are curious about which players are most likely to make a big impact on the league. Fans may learn more about the future of the league and the most interesting young players to watch by making accurate predictions. Finally, this study will reveal an essential aspect of understanding the sport of basketball itself. By identifying the factors that contribute to a player's success, we can gain a deeper understanding of the game and its nuances. We can learn about the importance of different factors, such as physical attributes, team dynamics, and individual statistics, in determining a player's future success.

In our first short report, we encountered the problem of skewness in the data as far as career outcomes. Furthermore, we could not precisely define a response variable to quantify the career success of a particular player. As a result, we decided to use the k-means method to generate three clusters, classifying a player's career into one of the three categories. It is important to note that this clustering was based on the player's career stats, not just their rookie seasons. We will then be able to use the resulting clustering of each player as a response variable and train various classification models based on the player's rookie statistics to predict the career outcome of the player.

2 Data Description

Our study will use data from Basketball Reference's website to analyze 829 NBA players who entered the league between 2005 and 2015. The dataset contains information on each player's rookie season and normal season, including statistics such as win shares, games played, points per game, field goal percentage, 2-point percentage, 3-point percentage, free throw percentage, true shooting percentage, effective field goal percentage, offensive rating, defensive rating, offensive win shares, defensive win shares, win shares per 48 minutes, offensive box plus/minus, defensive box plus/minus, box plus/minus, value over replacement player, and player efficiency rating.

The main statistics our study will focus on are offensive and defensive box plus/minus, offensive and defensive win shares, and VORP. VORP is a statistic that estimates a player's overall value compared to a replacement-level player. In simpler terms, it measures how much more valuable a player is compared to an average player who could easily be replaced.

Win shares is a statistic that attempts to quantify a player's contribution to team wins. It calculates the number of wins a player is responsible for based on their offensive and defensive production. Offensive win shares measure a player's offensive contributions, while defensive win shares measure their defensive contributions. Win shares per 48 minutes is a variation of win shares that measures a player's productivity over the course of 48 minutes.

Box plus/minus estimates a player's value above average, adjusted for position and league. It takes into account a player's offensive and defensive contributions, as well as their playing time. Offensive box plus/minus measures a player's offensive value above average, while defensive box plus/minus measures their defensive value above average. Box plus/minus is adjusted for the position played by the player and the overall level of play in the league.

To gather this data, we used Basketball Reference's premier Stathead package, which includes querying tools that were vital to obtaining the player data for this short report. These statistics will help us understand a player's overall value to the team, their offensive and defensive abilities, and how much more valuable they are compared to an average player who can easily be replaced.

3 Exploratory Data Analysis

In our exploratory data analysis, we explored the dataset and discovered that it was heavily right-skewed. When a distribution of data is heavily right-skewed, it means that the majority of the data values are concentrated on the left side of the distribution, and there are a few extremely large values on the right side that pull the mean of the distribution toward the right. In other words, the distribution has a long tail on the right side. In this case, the right-skew in our data is a result of the fact that most players didn't have successful NBA careers. When a distribution is heavily right-skewed, it can affect the interpretation of statistical measures such as the mean and standard deviation, as they may not

accurately represent the typical value or spread of the data. Therefore, it is important to take into account the skewness of the distribution when analyzing the data.

We faced another challenge in defining a response variable to quantify the career success of each player. Since NBA basketball is a complex sport with many variables that contribute to a player's success, it was difficult to determine a single variable that could accurately capture a player's overall performance. Therefore, we decided to use the k-means method to generate three clusters based on each player's career statistics, grouping players into one of three categories: 0: "bad," 1: "good," and 2: "mid."

To perform the clustering, we used the k-means function with 3 clusters and used all career variables in the dataset as features. After applying the k-means algorithm, we found that the three clusters were reasonably well separated, with one cluster containing mostly "bad" players, one containing mostly "mid" players, and one containing mostly "good" players. The cluster centroids can be interpreted as representing the typical career trajectory for each group of players.

4 Methodology

To generate a response variable that holistically represents a player's career success, we'll utilize K-means clustering. K-means clustering is a machine learning algorithm used for unsupervised learning tasks. The algorithm is based on the idea of dividing a set of observations into a predetermined number of clusters, K, in a way that minimizes the sum of squared distances between the observations and their corresponding cluster centers [4]. The algorithm starts by randomly selecting K cluster centers from the data points. Each observation is then assigned to the nearest cluster center based on its distance to that center. The cluster centers are then updated by calculating the mean of the observations assigned to each cluster, and the process is repeated until the cluster assignments no longer change or a maximum number of iterations is reached.

Once we've applied k-means clustering to generate a response variable for player career success, we'll fit the following models with rookie stats as predictors: neural networks, gradient boosting, random forest, and multinomial logistic regression.

Neural networks are a type of machine learning algorithm that consists of interconnected neurons. Each neuron processes input data and transmits its output to other neurons in the network. The structure of a neural network is inspired by the human brain, where each neuron receives input from neurons in the previous layer and produces output for neurons in the next layer. During the training phase, the network adjusts the weights and biases of its neurons to minimize the difference between predicted and actual output. Backpropagation is a process in which the network iteratively updates its weights and biases based on the error between predicted and actual output. Neural networks are effective at recognizing complex and nonlinear patterns in data, making them

valuable for image recognition, speech recognition, natural language processing, and game playing [2]. We used the TensorFlow and Keras libraries in Python to build a neural network model that would predict the cluster number for a given set of rookie statistics. Our neural network consisted of an input layer, four hidden layers with 256, 128, 64, and 32 neurons, respectively, and an output layer with three neurons. We trained our model on a randomly selected 80% of the dataset using the "sparse_categorical_crossentropy" loss function and the "adam" optimizer for 150 epochs, with a batch size of 32. Finally, we evaluated our model on the remaining 20% of the dataset to demonstrate the effectiveness of machine learning in sports analytics.

Gradient Boosting Machines (GBMs) are a type of machine learning algorithm that combines several weak models, such as decision trees, into a stronger, more accurate model [5]. The basic idea behind GBMs is to iteratively add new models to the ensemble in a way that each new model corrects the errors made by the previous ones. This is achieved by minimizing a loss function that measures the difference between the predicted and actual values of the target variable. One important hyperparameter for a GBM is the learning rate, which controls the contribution of each new model in the ensemble to the final prediction. A lower learning rate means that each new model contributes less to the final prediction, which can help prevent overfitting and improve the generalization performance of the model. However, it also means that the algorithm will take more iterations to converge to an optimal solution, which can make the training process slower. On the other hand, a higher learning rate means that each new model contributes more to the final prediction, which can make the algorithm converge faster but also increase the risk of overfitting. Therefore, it's important to carefully tune the learning rate to find the right balance between convergence speed and generalization performance. For our GBM, we iteratively tuned the learning rate to maximize the model accuracy on a validation data set.

Multinomial logistic regression is a statistical method used to analyze relationships between a categorical dependent variable with more than two categories and one or more independent variables [3]. The basic idea of multinomial logistic regression is to model the relationship between the dependent variable (also known as the response variable) and the independent variables (also known as predictor variables) using a logistic function. The logistic function is used to estimate the probability of each category of the dependent variable, given the values of the independent variables. In our case, our categories were the generated clusters based on NBA players' skills. In our model specifically, we used LabelEncoder to encode categorical target variables, LogisticRegression library to create the model, and metrics to evaluate the model performance. The LabelEncoder converts the categorical 'y' into a numeric format. The multi_class='ovr' parameter stands for "one-vs-rest", meaning that it will create a binary model for each class of the target variable to classify it as either that class or not that class, and the inverse_transform method of the LabelEncoder object to convert the predicted numeric 'y' values back into their original categorical format.

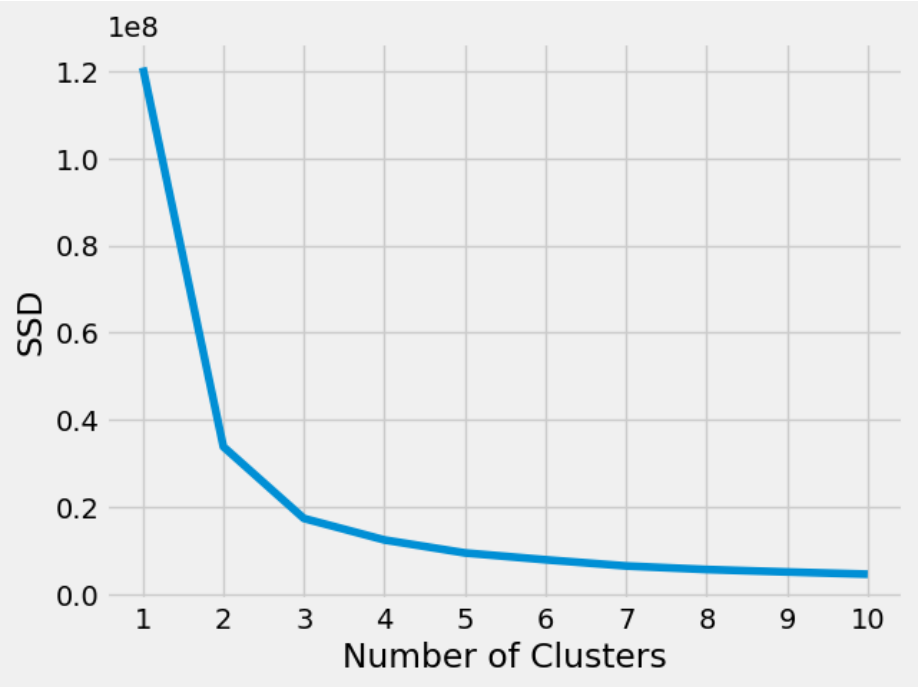
Random Forest is a supervised machine learning algorithm commonly used in

classification and regression. The algorithm works by building multiple decision trees on randomly sampled subsets of the data, then aggregating their predictions to make a final prediction. The random forest algorithm helps to reduce overfitting and improves the accuracy and stability of the model. The random forest algorithm also helps to reduce overfitting and improves the accuracy and stability of the model [1]. The objective of this model is to classify NBA players into one of three clusters based on their rookie season statistics. The code begins by importing the required libraries: 'sklearn.ensemble' for the random forest classifier, 'sklearn.model_selection' for train-test split and 'sklearn.metrics' for evaluation metrics such as accuracy, precision, recall, and F1-score. The data is then loaded into a dataframe and split into predictor variables and the target variable. The predictor variables contain the rookie season statistics of the players such as their age, field goal percentage, three-point percentage, free throw percentage, minutes per game, points per game, rebounds per game, assists per game, and rookie year. The target variable consists of three clusters generated in the K-means clustering. Next, the dataset is split into training and testing sets using the train_test_split function, with the testing set is twenty percent of the original data and a random seed of 42 being used to ensure reproducibility. A random forest classifier is then created using the RandomForestClassifier function from the sklearn.ensemble module. The hyperparameters of the model are specified, with 100 trees and a random seed of 42.

5 Results

5.1 K-Means Clustering

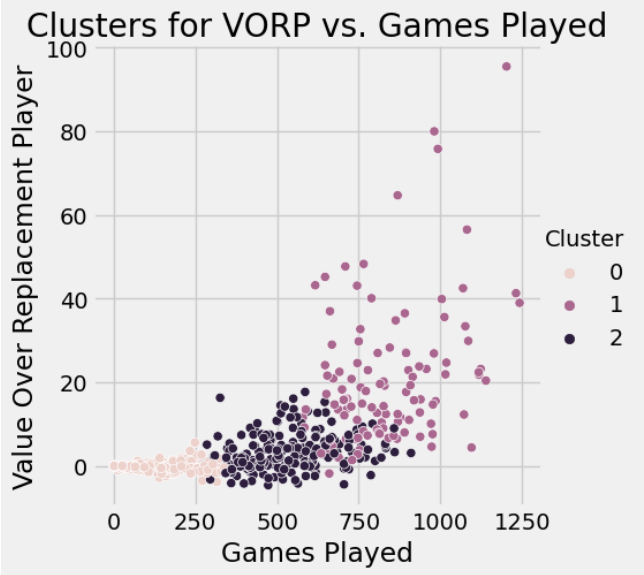
To generate a categorical response variable that captures the holistic career success of the players in our data set, we aimed to fit a k-means clustering model to all of the player's career statistics. In order to determine the optimal number of clusters for our data, we used the elbow method for the sum of squared distances in our clusters. Specifically, this SSD represents the sum of squared distances between each point in a cluster and that cluster's centroid.

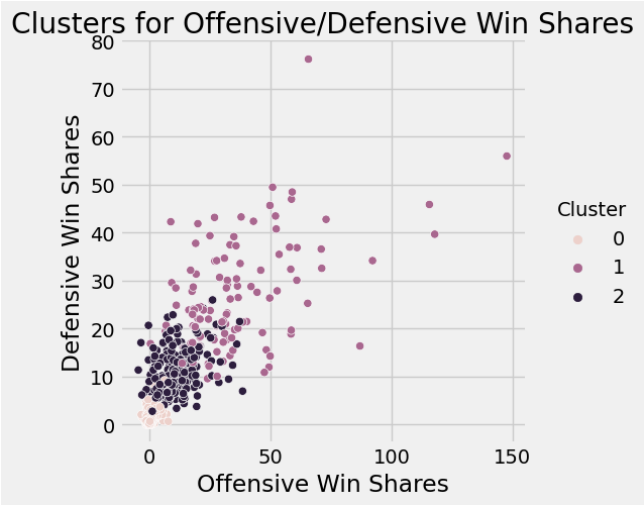


Going off of this plot, we chose to fit a 3-cluster k-means model to our data since there is a minimal decrease in SSD for models with more than 3 clusters. Before clustering, our hope was that the algorithm would group players by their overall career success (i.e. one cluster of star players, one cluster of average players, and one of the unsuccessful players). As we'll see in the following tables and plots, it appears that we've achieved that goal. The table below shows many different career statistics, averaged by cluster.

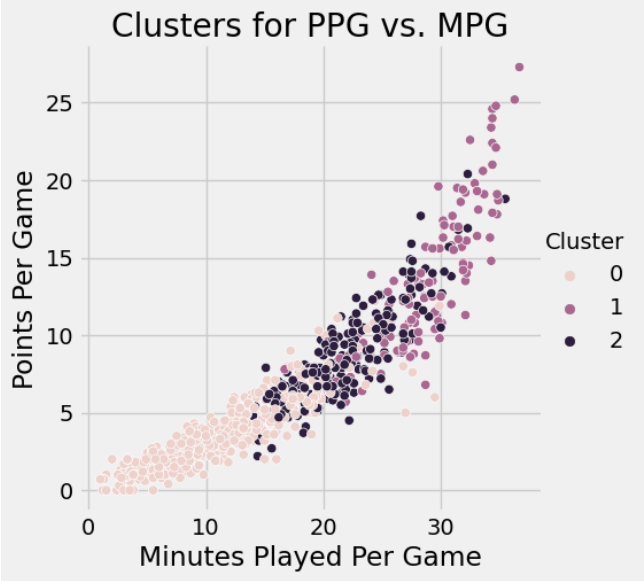
Cluster	Number of Players	Win Shares	Games Played	Points Per Game
0	500	1.3246	86.442	3.46
1	115	62.6973913	836.5217391	13.76782609
2	209	21.38038278	529.1913876	8.546889952
Cluster	**Field Goal Percent**	**2 Point Percent**	**3 Point Percent**	**Free Throw Percent**
0	0.405696	0.435858	0.200184	0.610606
1	0.469434783	0.502695652	0.32206087	0.769269565
2	0.457722488	0.490028708	0.290645933	0.74869378
Cluster	**True Shooting Percent**	**eFG Percent**	**Offensive Rtg**	**Defensive Rtg**
0	0.469656	0.436626	94.47	108.008
1	0.557356522	0.517634783	110.6869565	107.4695652
2	0.540555024	0.506406699	107.4784689	108.3253589
Cluster	**Offensive Win Shares**	**Defensive Win Shares**	**Win Shares Per 48**	**Offensive BPM**
0	0.3056	1.0194	0.016656	-3.611
1	35.90434783	26.79478261	0.121565217	0.951304348
2	10.63062201	10.7507177	0.088851675	-0.857416268
Cluster	**Defensive BPM**	**BPM (Box Plus Minus)**	**VORP**	**PER**
0	-0.8022	-4.413	-0.2096	9.5384
1	0.199130435	1.147826087	20.46347826	16.81652174
2	-0.132535885	-0.992822967	3.367464115	13.57655502

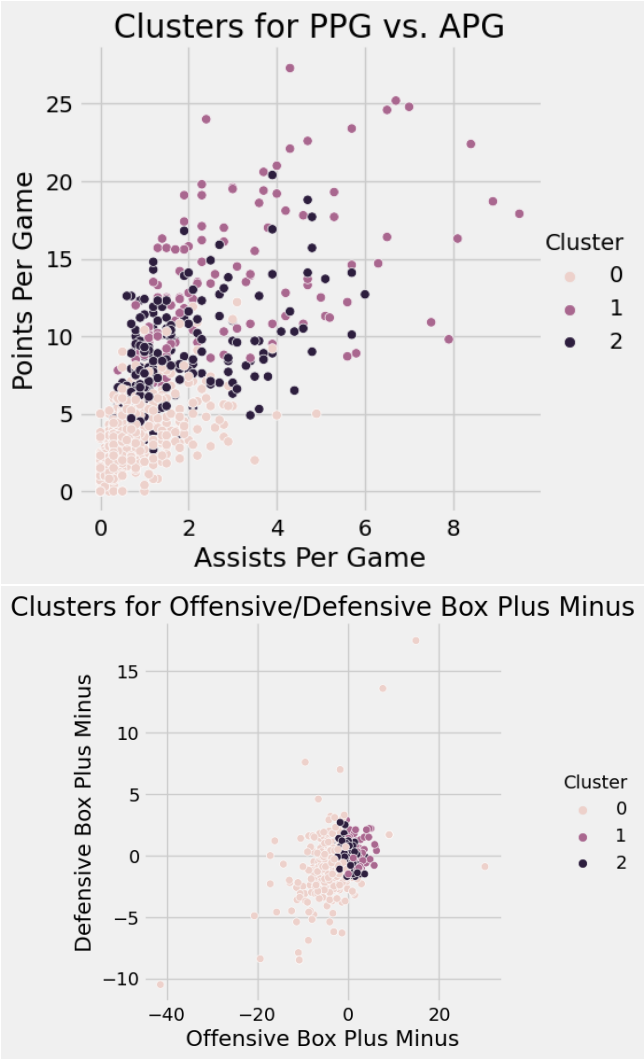
We can see that Cluster 0 has the most players and the lowest values in most statistics, Cluster 1 has the least players and the highest values in most statistics, and Cluster 2 is in between. In particular, we see a lot of differentiation between clusters in statistics that reward career longevity, such as Win Shares, Games Played, and VORP (Value Over Replacement Player). Since longevity is a direct result of player success, this is a good indication that our clustering has produced a valuable response to our situation. The following plots will illustrate how our clusters grouped players by a variety of features in our data set.





Both of these plots illustrate statistics that are cumulative for a player's career: VORP, games played, and offensive and defensive win shares. Players with long, successful careers will have high values for these stats, and players who didn't last in the league will be close to zero. Between these two plots, we can clearly see that our k-means algorithm has clustered players effectively according to these features.





These three plots display features that capture a player's effectiveness on a per-game basis: points per game, minutes per game, assists per game, and offensive and defensive BPM. Here we see that our clustering algorithm has also done a great job of separating good, average, and bad players according to these efficiency statistics.

Since k-means clustering has proven effective at grouping players by both statistics that capture career longevity and efficiency, we've accomplished our goal of generating a categorical response variable that holistically captures a player's level of career success. With this, we'll be able to move forward to the next phase of our analysis, where we'll fit a variety of classification models with rookie season stats as the feature variables and cluster as the response. These classification models will allow us to understand to what extent NBA career

success can be predicted purely off on a player's rookie season.

5.2 Classification Modeling

Using career success clusters as the response variable and rookie season statistics as the predictor variables, four classification models were fitted: neural network, gradient boosting, random forest, and multinomial logistic regression.

	Neural Network	Gradient Boosting	Random Forest	Multinomial Logistic
Accuracy	0.71	0.70	0.68	0.68
Cluster 0 Recall	0.89	0.93	0.90	0.94
Cluster 1 Recall	0.08	0.31	0.36	0.33
Cluster 2 Recall	0.63	0.36	0.37	0.26
Cluster 0 Precision	0.81	0.79	0.74	0.74
Cluster 1 Precision	1.00	0.48	0.64	0.58
Cluster 2 Precision	0.46	0.46	0.47	0.45

Table 1: Classification Results

In order to interpret our model results, it's important to understand two key terms: recall and precision. Recall measures the proportion of correctly classified positive instances among all actual positive instances in the data. In other words, it measures the sensitivity of the model to positive instances. Precision, on the other hand, measures the proportion of correctly classified positive instances among all instances predicted as positive. In other words, it measures the accuracy of positive predictions. In general, precision and recall are trade-offs, meaning that improving one may come at the cost of reducing the other. We see these trade-offs in our model results, and choosing the best model for our data will depend on how we view these trade-offs in the context of our question of interest.

In general, we can see that our models tend to have higher precision than recall for clusters 1 and 2 (with the exception of neural network's performance on cluster 2). This is likely due to an imbalance in our data, as players with successful NBA careers are a relative rarity. Thus, models are incentivized to be conservative in predicting career success to maximize accuracy.

Looking at our model results, we can see that our neural network has the highest overall accuracy, with our GBM in second place and random forest and multinomial logit tied for third. However, when we dig into the recall and precision data by cluster, we can see that accuracy doesn't tell the whole story of each model's performance. Our neural network had a cluster 1 recall value of 0.08 and a cluster 1 precision value of 1. This is because it only predicted two players in the testing data to have cluster 1 careers, and was correct for both. Essentially the model maximized accuracy by overwhelmingly predicting high-performing rookies to have cluster 2 careers, resulting in a much higher cluster 2 recall than other models. However, since the goal of our research is to predict future NBA stars (i.e., cluster 1 players), we'll want a model with

a much higher cluster 1 recall. We can see from the table that the random forest model has the highest cluster 1 and second highest cluster 2 recall values without sacrificing much accuracy. Thus, we'll select the random forest model to make predictions on present-day NBA rookies.

5.3 Present Day Predictions

We use the random forest model to generate predictions of how well this year's rookie class would pan out.

Name	Draft Number	Predicted Cluster	All-Rookie Team
Paolo Banchero	1	1	First
Jabari Smith Jr.	3	1	Second
Keegan Murray	4	2	First
Jaden Ivey	5	2	Second
Bennedict Mathurin	6	1	First
Shaedon Sharpe	7	2	N/A
Dyson Daniels	8	2	N/A
Jeremy Sochan	9	2	Second
Johnny Davis	10	2	N/A
Malaki Branham	20	1	N/A
RaiQuan Gray	Undrafted	1	N/A
AJ Griffin	16	1	N/A
Jaden Hardy	37	1	N/A
Walker Kessler	22	1	First
Jaylin Williams	34	1	N/A
Tari Eason	17	2	Second
Jalen Williams	12	2	First
Jalen Duren	13	2	Second

Table 2: Predicted clusters for NBA draft prospects

The majority of the top 10 draft picks were clustered as either category 1 or 2, indicating that no draft "busts" were selected. This is an encouraging outcome for team managers and scouts, as it suggests that they did not select any individuals with a high likelihood of underperforming in the NBA. When comparing our clustering results to the All-NBA rookie first and second teams, our model demonstrated a high degree of success in accurately predicting which players would become either highly successful or moderately successful in the NBA. However, there were some players, including Jalen Williams, Jalen Duren, Jaden Ivey, Jeremy Sochan, and Keegan Murray, who were voted by NBA coaches as among the ten best rookies of the season, but our model categorized as moderately successful instead of highly successful. This may be attributed to these players' tendency to generate less impressive stat lines compared to other players, which may have been weighted more heavily in our model's algorithm.

6 Conclusion

Using data from Basketball Reference, four classification models (neural network, gradient boosting, random forest, and multinomial logistic regression) were fitted to predict NBA players' career success from their rookie season statistics alone. The data for this research consisted of the rookie season and total career stats for all NBA players who entered the league from 2005-2015. K-means clustering was used to group players by total career stats, and the clustering output was used as our response variable for classification modelling. Three clusters were generated: one which represented players with excellent NBA careers, one for "role players" who lasted in the league but weren't stars, and one for players who did not make it long in the NBA. Models were compared by their accuracy, as well as precision and recall for each cluster. Models were generally conservative in predicting player success to maximize accuracy. Due to its comparatively high recall for successful players, the random forest model was selected and used to predict career outcomes for present day rookies. There are many players who grow into successful careers after poor rookie seasons, and we don't expect our model to detect those players with much accuracy. However, we believe that this methodology will yield high-precision results for this rookie class and the future, and can provide useful insight to front offices, media, and fans who want to predict the career trajectories of rookie players.

References

[1] Gérard Biau and Erwan Scornet. "A random forest guided tour". In: *Test* 25 (2016), pp. 197–227.

[2] Chris M. Bishop. "Neural networks and their applications". In: *Review of Scientific Instruments* 65.6 (June 1994), pp. 1803–1832. ISSN: 0034-6748. DOI: 10.1063/1.1144830. eprint: https://pubs.aip.org/aip/rsi/article-pdf/65/6/1803/8387807/1803_1_online.pdf. URL: https://doi.org/10.1063/1.1144830.

[3] Dankmar Böhning. "Multinomial Logistic Regression Algorithm". In: *Annals of the Institute of Statistical Mathematics* 44.1 (1992), pp. 197–200. URL: https://EconPapers.repec.org/RePEc:spr:aistmt:v:44:y:1992:i:1:p:197-200.

[4] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346830 (visited on 05/09/2023).

[5] Alexey Natekin and Alois Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in Neurorobotics* 7 (2013). ISSN: 1662-5218. DOI: 10.3389/fnbot.2013.00021. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021.

7 Appendix: Source Code (Python)

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import warnings
warnings.simplefilter('ignore')
from sklearn.cluster import KMeans
import numpy as np
import seaborn as sns
from google.colab import files
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

    df = pd.read_csv('https://raw.githubusercontent.com/grantwestfall/stat_482_p
dfRookie = pd.read_csv('https://raw.githubusercontent.com/justiincho/STAT482Data

    df[["3P_Percent", "FT_Percent", "FG_Percent", "2P_Percent", "eFG_Percent", "
df[["3P_Percent", "FT_Percent", "FG_Percent", "2P_Percent", "eFG_Percent", "TS_P

dfRookie[["Rook_3P_Percent", "Rook_FT_Percent", "Rook_FG_Percent"]] = \
dfRookie[["Rook_3P_Percent", "Rook_FT_Percent", "Rook_FG_Percent"]].fillna(0)
df = df.dropna()
career_stats = df[['WS', 'G', 'GS',
       'MP', 'FG', 'FGA', '2P', '2PA', '3P', '3PA', 'FT', 'FTA', 'ORB', 'DRB',
       'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'FG_Percent',
       '2P_Percent', '3P_Percent', 'FT_Percent', 'TS_Percent', 'eFG_Percent', 'C
       'DBPM', 'BPM', 'VORP', 'PER', 'ORB_Percent', 'DRB_Percent',
       'TRB_Percent', 'AST_Percent', 'STL_Percent', 'BLK_Percent',
       'TOV_Percent', 'USG_Percent']]

    from sklearn.cluster import KMeans
kmeans_kwargs = {
       "init": "random",
       "n_init": 10,
       "max_iter": 300,
       "random_state": 42,
   }

kmc = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300,tol=1e-04, ran
y_kmc = kmc.fit_predict(career_stats)
ssd = []
for k in range(1, 11):
        kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
```

```
        kmeans.fit(career_stats)
        ssd.append(kmeans.inertia_)

plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), ssd)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSD")
plt.show()

    kmeans = KMeans(n_clusters=3, random_state=0, n_init="auto").fit(career_stat
    df["Cluster"] = kmeans.labels_
    cluster_summary = df.groupby(by = "Cluster").mean()[['WS', 'G','PTS', 'FG_Pe
        '2P_Percent', '3P_Percent', 'FT_Percent', 'TS_Percent', 'eFG_Percent',
        'ORtg', 'DRtg', 'OWS', 'DWS', 'WS_Per_48', 'OBPM',
        'DBPM', 'BPM', 'VORP', 'PER']]

    sns.relplot(data = df, x = "OWS", y = "DWS", hue = "Cluster").set(xlabel = "
                                                                    title = "Clust
sns.relplot(data = df, x = "G", y = "VORP", hue = "Cluster").set(xlabel = "Games
                                                                  title = "Cluste
sns.relplot(data = df, x = "OBPM", y = "DBPM", hue = "Cluster").set(xlabel = "Of
                                                                    title = "Clu
sns.relplot(data = df, x = "AST", y = "PTS", hue = "Cluster").set(xlabel = "Assi
sns.relplot(data = df, x = "MP", y = "PTS", hue = "Cluster").set(xlabel = "Minut

    import numpy as np
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dropout

    X = df[['Rook_Age', 'Rook_FG_Percent', 'Rook_3P_Percent',
        'Rook_FT_Percent', 'Rook_MPG', 'Rook_PPG', 'Rook_RPG', 'Rook_APG', 'Rook_
Y = df['Cluster']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, rando

    X_train_2, X_val, Y_train_2, Y_val = train_test_split(X_train, Y_train, test

lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_
    gb_clf.fit(X_train_2, Y_train_2)
```

```
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train_2, Y_
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, Y_va

xgb_clf = XGBClassifier()
xgb_clf.fit(X_train_2, Y_train_2)
print(f"XGBoost score (validation): {xgb_clf.score(X_val, Y_val)}")

gb_clf_opt = GradientBoostingClassifier(n_estimators = 20, learning_rate = 0.25,
gb_clf_opt.fit(X_train_2, Y_train_2)
print(f"Optimized model accuracy score (testing): {gb_clf_opt.score(X_test, Y_te

predictions = gb_clf_opt.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(Y_test, predictions))

print("Classification Report")
print(classification_report(Y_test, predictions))

    from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

X1 = df[['Rook_Age', 'Rook_FG_Percent', 'Rook_3P_Percent',
       'Rook_FT_Percent', 'Rook_MPG', 'Rook_PPG', 'Rook_RPG', 'Rook_APG', 'Rook_
Y1 = df['Cluster']

X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size=0.2, rando


rfc = RandomForestClassifier(n_estimators=100, random_state=42)

rfc.fit(X_train, y_train)


y_pred = rfc.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
classification = classification_report(y_test, y_pred)
```

```
print("Random Forest Accuracy:", accuracy)
print("Random Forest Precision:", precision)
print("Random Forest Recall:", recall)
print("Random Forest F1 Score:", f1)
print("Random Forest Mean Squared Error:", mse)
print("Random Forest R-squared Score:", r2)
print("Random Forest Confusion Matrix:")
print(cm)
print("Random Forest Classification Report:")
print(classification)

    rookieXPred = dfRookie[['Rook_Age', 'Rook_FG_Percent', 'Rook_3P_Percent',
       'Rook_FT_Percent', 'Rook_MPG', 'Rook_PPG', 'Rook_RPG', 'Rook_APG', 'Rook_
rookieCluster = rfc.predict(rookieXPred)

dfRookie['Predicted_Cluster'] = rookieCluster
dfRookie
dfRookie[dfRookie["Predicted_Cluster"] == 1]

    import numpy as np
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.layers import Dropout

X1 = df[['Rook_Age', 'Rook_FG_Percent', 'Rook_3P_Percent',
       'Rook_FT_Percent', 'Rook_MPG', 'Rook_PPG', 'Rook_RPG', 'Rook_APG', 'Rook_
Y1 = df['Cluster'].astype(int)



x_train, x_test, y_train, y_test = train_test_split(X1, Y1, test_size=0.2, rando

neuralNetwork = Sequential()
neuralNetwork.add(Dense(256, input_shape = (9,), activation = 'relu'))
neuralNetwork.add(Dense(128, activation = 'relu'))
neuralNetwork.add(Dense(64, activation = 'relu'))
neuralNetwork.add(Dense(32, activation = 'relu'))
neuralNetwork.add(Dense(3, activation = 'sigmoid'))

neuralNetwork.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam
neuralNetwork.fit(x_train, y_train, epochs = 150, batch_size = 32)

print()
```

```
neuralNetwork.evaluate(x_test, y_test)

    x_test_prediction = neuralNetwork.predict(x_test)
x_test_pred = np.argmax(x_test_prediction, axis = 1)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("Classification Report")
print(classification_report(y_test, y_pred))

print(f'The accuracy is: {accuracy}.')
print(f'The precision is: {precision}.')
print(f'The recall is: {recall}.')

    from sklearn.datasets import make_classification
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

X = df[['Rook_Age', 'Rook_FG_Percent', 'Rook_3P_Percent',
        'Rook_FT_Percent', 'Rook_MPG', 'Rook_PPG', 'Rook_RPG', 'Rook_APG', 'Rook_
y = df['Cluster']

encoder = LabelEncoder()
y_enc = encoder.fit_transform(y)

model = LogisticRegression(multi_class='ovr')
model.fit(X, y_enc)

y_pred = model.predict(X)

y_pred_dec = encoder.inverse_transform(y_pred)

accuracy = accuracy_score(y, y_pred_dec)
precision = precision_score(y, y_pred_dec, average='weighted')
confusion_mat = confusion_matrix(y, y_pred_dec)
classification = classification_report(y, y_pred_dec)


print("Logistic Regression Accuracy:", accuracy)
print("Precision:", precision)
print("Confusion Matrix:\n", confusion_mat)
```

```
print(classification)
```