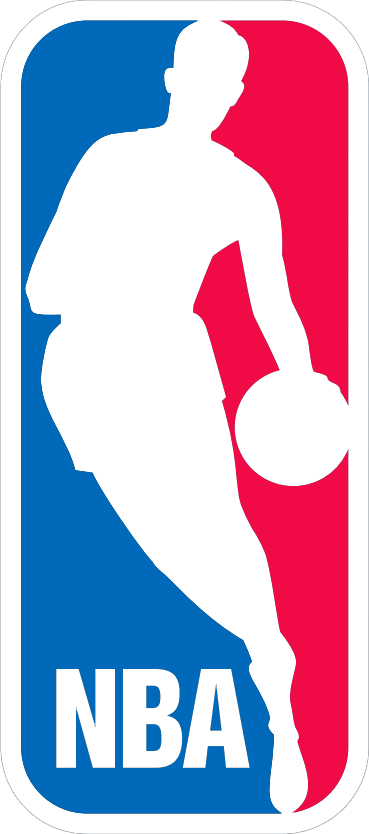
**Advanced Analytics in Basketball Performance Prediction**

Using advanced NBA statistics to predict All Star appearances





FINAL PROJECT DSO499

Name: **Drew Osherow**

Student ID: **6130663389**

Email: **dosherow@usc.edu**

**Abstract**

After studying recent articles and publications on building predictive models for the NBA, it was interesting to note that with such an influx in advanced statistics and tracking in recent year for the sport, no studies utilized all of the metrics that are available for predicting All Star appearances. The All Star game in the NBA happens once per year normally in the middle of February which marks the mid-way point in the season (even though teams have played more than 60% of their total games). It is a collection of the best players during the course of each season playing one game against one another regardless of their employed team. The selection process for NBA All Stars has changed throughout recent years, in the early 2000’s fans were able to vote in the NBA arenas during games and online through ballots the starters for the game which ends up being ten players, five for either team. To fill out the remainder of the rosters, coaches from each team in the NBA select the seven reserve players for each team. In recent years, the two players who received the most fan votes have been selected as team captains and in turn they draft players for each of their respective All Star teams and again coaches from around the league fill out the remainder of the rosters. These methods of selection provide a good mix of popularity factor biasing fans (players who have been around for more years who are more well-known are likely to be voted more often by fans then younger players who just entered the league), and coaches/players themselves evaluating their peers in their respective profession.

The goal of this research is to build multiple predictive models using deep learning and machine learning algorithms to find the highest accuracy in predicting correctly if a player will become an All Star or not based on a wide variety of advanced statistics that are recorded during NBA regular season games. Of these statistics, some have been around for years, are simple to calculate, and are a part of every single games box score which is a summary of statistics for each team every game played. Examples of these statistics include Field Goals Attempted, Field Goals Made, Three Pointers Attempted, Three Pointers Made, Rebounds, Assists, and Points. Advanced statistics that have become a huge part of team building from an executive standpoint and player acquisition standpoint have led many teams to rethinking how they scout players in the league and what they focus on doing in games. These statistics are usually calculated from a combination of the more simple statistics. Some of these statistics include Player Efficiency Rating, True Shooting Percentage, Usage Percentage, and Win Shares. For an idea of the metrics that go behind some of these more advanced statistics here is a breakdown of the formula for Player Efficiency Rating.

**PER =** (1 / MP) \*

[ 3P

+ (2 / 3) \* AST

+ 2 – factor \* (team\_AST / team\_FG)) \* FG

+ (FT \* 0.5 \* (1 + (1 – (team\_AST / team\_FG)) + (2 / 3) \* (team\_AST / team\_FG)))

* VOP \* TOV
* VOP \* DRB% \* (FGA – FG)
* VOP \* 0.44 \* (0.44 + (0.56 \* DRB%)) \* (FTA – FT)

+ VOP \* (1 – DRB%) \* (TRB – ORB)

+ VOP \* DRB% \* ORB

+ VOP \* STL

+ VOP \* DRB% \* BLK

* PF \* ((lg\_FT / lg\_PF) – 0.44 \* (lg\_FTA / lg\_PF) \* VOP) ]

PER sums up all a player’s positive accomplishments, subtracts the negative accomplishments, and returns a per-minute rating of a player’s performance. Just looking at the formula for this one individual stat, it will be interesting to see how the combination of factors weigh more or less on predicting the likelihood of a player becoming an All Star as opposed to them individually. For the purpose of this research, we are only interested in more recent years in the NBA. Our data spans from 2000-2017. The game of basketball in the NBA has shifted tremendously in recent years, with a change in focus on Three Pointers and Free Throws as opposed to long Two Pointers and other shots with low expected value. The goal is to find the model that has the highest accuracy in predicting whether a player will be an All Star or not from a combination of these statistics. Next I will focus on the data collection and pre-processing for building the models.

**Data Collection**

I downloaded a Kaggle dataset from the user Omri Goldstein for this research. The spreadsheet I used had fifty three columns and over twenty four thousand and seventy rows. The original data contained statistics from the NBA for every player in the league from 1950-2017. For the purpose of this research and due to time constraints we focused on only the years from 2000-2017.

**Pre-Processing Data and Feature Engineering**

One thing I noticed immediately in the dataset was that there were a lot of duplicate players during each season. This shouldn’t be the case as each unique player should only have one row of statistics for that season. I realized that due to trades and players getting released and signed to other teams that if a player had a value for the column Team that said TOT, that meant they had moved teams during the season. To account for this I manually deleted every row for duplicate players under the TOT row because these rows were just the summation of the TOT row for the full season. I also noticed that there were two blank columns so I deleted those as well. Next, using Wikipedia for each seasons All Star game, I created a target variable column at the end of the dataset which would take binary values in a 0 if a player wasn’t an All Star that year and a 1 if the player was. After doing this, I created another column before the target variable which would account for Previous All Star appearances. This column would signify the power of popularity for the fans as players with multiple Previous All Star appearances would stand out more in the voting process. I was interested to see how the models would weigh this variable and if it would serve to be a significant factor in predicting All Stars. This process took quite some time as I had to go back the previous decade and manually input both players towards the end of their careers in the early 2000’s to see how many times they were All Stars before the range of dates we were focusing on. After doing this, it seemed like the data was ready to be imported into the production process for building the models. Before doing so, I thought about some other factors in the data that may skew the results and prediction. I decided to put an emphasis on players who had played over a certain number of games in the season and who had played over a certain number of minutes in the season to proceed forward with in building the models. My thought process was that since we are running a classification, the True Negatives in the prediction process may severely skew the prediction model to be more accurate then it may truly be. During the course of the NBA season when you think about it, if a team rosters fifteen players and only gives significant minutes to about nine or ten of them, with six through ten receiving significantly minutes less than the starting five, these players will easily be predicted to not be all stars since they have such a small sample size. Also, injuries are a normal part of the NBA season and happen at random, but in the context of a dataset, if Player A plays in twenty three games but logs only five minutes per game, and Player B plays in only five games but logs thirty five minutes per game, we can tell that Player B was a significant contributor and most likely got injured just based off the minutes alone. To account for the possible skew in games and minutes, I set a cut-off for only accounting for players who played at least forty games during the regular season as the regular season contains eighty two total games per team. Sine the All Star game is about how way through the season this should help account for early injuries from star players and players who don’t get many opportunities to find the court during games. I also set a cut-off of a minimum one thousand two hundred minutes played to help account for players who got very few minutes but played in a majority of games during the season. After performing this data cleansing, the final data set to be used for the prediction models contained four thousand one hundred and eighteen rows including header row, fifty one explanatory variables (statistics), and one target variable.

**Choosing Models and Setting Up Environment**

Our problem is a classification problem. We are looking to predict from a variety of explanatory variables whether a player will be an All Star or not. There are many different models to choose when trying to find the best accuracy in prediction for classification and I have selected four for the purpose of this study. The ones I selected are Logistic Regression, Random Forest, XGBoost, and MLP. I used Python to run the various models and input and output the data for visualization and analysis. First I will go over setting up the environment for running the models in PyCharm. First we will load up our environment with the dependencies we need to run the code.

from \_\_future\_\_ import absolute\_import  
from \_\_future\_\_ import division  
from \_\_future\_\_ import print\_function  
  
import numpy as np  
from keras.models import Sequential  
from keras.layers import Dense  
import pandas as pd  
import matplotlib.pyplot as plt  
import os  
from pandas import DataFrame  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn import metrics  
from sklearn import tree  
import xgboost as xgb  
from sklearn.metrics import confusion\_matrix  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report

After loading up the dependencies we will use in this study, I now load the csv file that has been cleansed and ready to analyze. I store the file as a Pandas DataFrame and create a variable to hold the tabular data and keep formatting how it was on the spreadsheet. Next I check to see if there are any null values in any of the rows in the dataset which will be necessary to fix in order to run some of the models and not bias the results in any way. To do this, I search the DataFrame for null values and print count of non-null values for each column to the terminal. I see that every column except 3 Point Percentage has four thousand one hundred and seventeen non-null values. To alleviate the missing two hundred and forty one values which is due to some players not attempting any 3 Pointers during the season, I create a function that fills null values in 3 Point Percentage with the median value of that column. This will create less of an issue later on as if we were to set every null value to 0, players who are stars in multiple categories yet don’t shoot any 3 Pointers would be penalized.

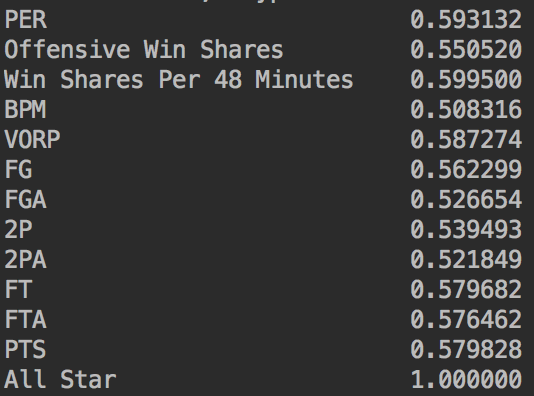
#open csv file  
stats = pd.read\_csv('Seasons\_Stats 2.csv')  
  
#store table as a dataframe  
df = DataFrame(stats)  
print(df.info())  
  
#checking for null values that might skew model  
null\_values = df.isnull().sum()  
null\_values = null\_values[null\_values !=0].sort\_values(ascending=False).reset\_index()  
null\_values.columns = ['variable', 'number of missing']  
print(null\_values)  
  
#using median value of each column to fill null values  
  
def fillWithMedian(data):  
 return data.fillna(data.median(), inplace=True)  
  
fillWithMedian(df)  
df.isnull().any()  
print(df.isnull().any())  
print(df.info())

Next, I want to check for correlation between explanatory variables and the target variable to see if we can hone in on the variables used for fitting the models. In the long run, using less variables should better account for overfitting of our prediction accuracy. I create a correlation matrix on the DataFrame then split the data into features and target variables for the remainder of the analysis. I remove the Year column from the data for analysis as we will randomize the training and testing samples later on, as well as remove the Player Name, Team, and Position columns. I do this because these are categorical data values that are non-numeric and to perform label encoding on this study isn’t in our best interest for what we’re hoping to accomplish. We don’t want to focus on positions because an equal number of players at each position are selected every year which is consistent for the All Star Game, and we don’t want to include the Team because we are focused on individual stats. After running the feature importance methods, we see that Win Shares Per 48 Minutes, Player Efficiency Rating, Value Over Replacement Player, Points, Free Throws Made, Free Throws Attempted, Field Goals Made, Offensive Win Shares, 2 Pointers Made, Field Goals Attempted, and 2 Pointers Attempted showed the most significance. It’s interesting to note at this point that some of the most significant factors before testing are the advanced statistics such as PER, VORP, and Win Shares Per 48 Minutes. Also, it can be noted that offensive metrics heavily outweigh significance over defensive metrics. This can be attributed to the game of basketball being a form of entertainment, and high octane, high-scoring games are more exciting for fans than low scoring defensive battles. Placing a premium on high scoring and offensive efficiency is extremely lucrative for the NBA and attracting viewership.

#check for correlation between variables and target  
  
corr\_matrix = df.corr()  
print(corr\_matrix['All Star'].sort\_values(ascending=False))  
  
#split dataset into features and target variable  
X = df[['Age', 'Games', 'Games Started', 'Minutes Played',  
 'PER', 'TS%', '3P Attempt Rate', 'FT Rate', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%',  
 'BLK%', 'TOV%', 'USG%', 'Offensive Win Shares', 'Defensive Win Shares', 'Win Shares Per 48 Minutes',  
 'Win Shares Per 48 Minutes.1', 'OBPM', 'DBPM', 'BPM', 'VORP', 'FG', 'FGA', 'FG%', '3P', '3PA',  
 '3P%', '2P', '2P%', 'eFG%', 'FT', 'FT%', 'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF',  
 'PTS', 'Prev All Star']]  
y = df['All Star']  
  
  
#feature selection  
cor = df.corr()  
cor\_target = abs(cor['All Star'])  
  
#feature importance  
#selecting highly correlated features  
relevant\_features = cor\_target[cor\_target>0.5]  
print(relevant\_features)

Output:

**Overview of Predictors: 10 Most Significant Predictors:**

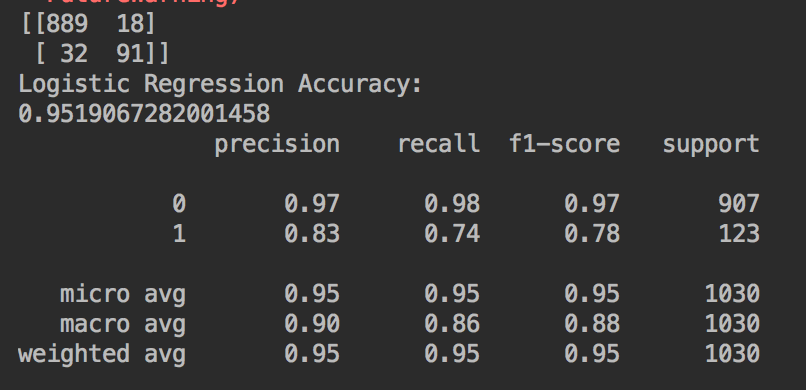


**Predictions and Analysis**

First I run the logistic regression using the baseline with all explanatory variables included in the model. I split the training/testing into a 75%/25% split and input a random state. The resulting accuracy score was an astounding 0.9519 on the baseline alone.

#split X and y into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  
  
log\_reg = LogisticRegression()  
  
# fit model with data  
  
log\_reg.fit(X\_train, y\_train)  
  
# make predictions  
y\_pred = log\_reg.predict(X\_test)  
  
# evaluate performance  
confusion = metrics.confusion\_matrix(y\_test, y\_pred)  
print(confusion)  
print("Logistic Regression Accuracy:")  
print(log\_reg.score(X,y))  
print(classification\_report(y\_test, y\_pred))

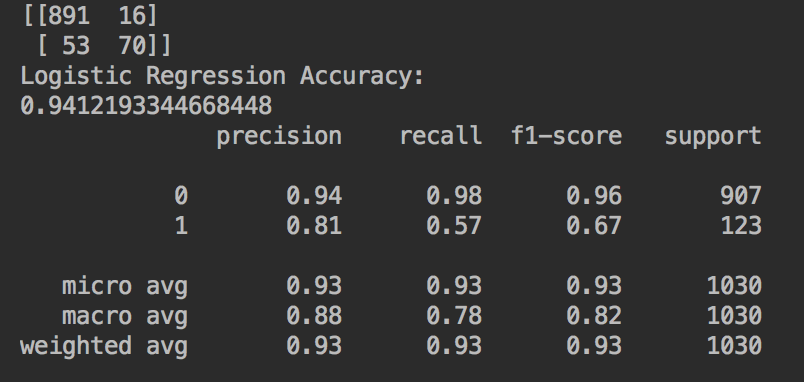
Output:



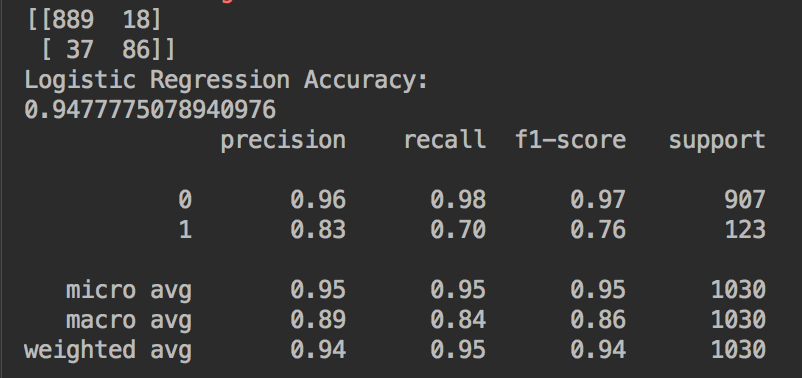
After running the baseline, I chose to remove every variable except for the top fifteen from the confusion matrix to see how the prediction accuracy would matchup.

#drop irrelevant variables for better model performance  
col\_remove = df[['Games', 'Games Started', 'Minutes Played', 'TS%', '3P Attempt Rate', 'FT Rate',  
 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%', 'BLK%', 'TOV%', 'Defensive Win Shares',  
 'DBPM', 'FG%', '3P%','2P%', 'eFG%', 'FT%', 'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'PF',  
 'Prev All Star']]  
X1 = X.drop(col\_remove, axis=1)  
# print(X1.info())

Output:



I was surprised to see that the prediction accuracy dropped quite a bit to 0.9412. The baseline was able to accurately predict twenty one more True Positives than the filtered featured selected one. I thought about why this could be and I figured it may have something to do with removing Games, Games Played, and Minutes. It’s interesting to note that the feature selected model had fewer False Positives than the baseline, although not by few, but the real difference maker came in the baseline’s performance of False Negatives. The baseline had way fewer of these instances which was the driving force behind the better performance. After putting Games, Games Played, and Minutes back into the model, I ran the code once more and we came up with a accuracy score way closer to the baseline.

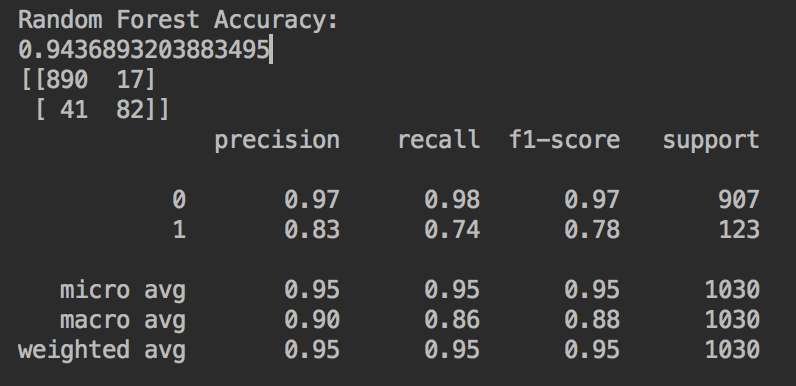


Since we are interested in labeling True Positives to the greatest accuracy and since a majority of our data will be assigned to True Negatives due to the number of elite players and selection spots for the All Star game, we should focus on the confusion metrics reflecting the True class. The baseline model had the highest F1-Score which takes into account the weighted average of Precision and Recall. Since we have seen that False Negatives hurt us more than False Positives, we rely on both accuracy and F1-Score. Because of the flexibility of the baseline, I decide to utilize it for the remainder models. The next model I choose to run is a Random Forest Classifier.

The Random Forest Classifier builds multiple decision trees and merges them together to get a more accurate and stable prediction. Advantages of Random Forest over classic Decision Trees is that instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. Only a random subset of features is taken into consideration by the algorithm for splitting nodes. Some hyperparameter tuning I did for the classifier was set n\_estimators=40 and max\_features=10 which I saw provided the biggest increase in performance over the baseline. n\_estimators is the number of trees the algorithm builds before deciding on predictions. max\_features is the maximum number of features the algorithm considers to split a node.

## ---------------- RANDOM FOREST ------------------- ##  
  
model = RandomForestClassifier(n\_estimators=40, max\_features=10)  
model.fit(X\_train, y\_train)  
print("Random Forest Accuracy:")  
print(model.score(X\_test,y\_test))  
y\_predicted = model.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_predicted)  
print(cm)  
print(classification\_report(y\_test, y\_pred))

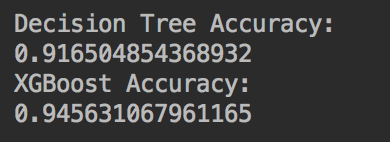
Output:



Next, I wanted to use a very popular algorithm called XGBoost which is relatively new in machine learning. XGBoost stands for Extreme Gradient Boosting and implements the gradient boosting decision tree algorithm. This model inparticular is quick in computations and has a lot of great features for tuning hyperparameters.

training\_data = X\_train  
test\_data = X\_test  
  
training\_target = y\_train  
test\_target = y\_test  
  
our\_tree = tree.DecisionTreeClassifier()  
  
our\_tree.fit(training\_data,training\_target)  
  
weak\_accuracy\_test = our\_tree.score(test\_data,test\_target)  
  
print("Decision Tree Accuracy:")  
print(weak\_accuracy\_test)  
  
our\_xgbooster = xgb.XGBClassifier(objective='binary:logistic', colsample\_bytree= 0.25, learning\_rate = 0.1,  
 max\_depth=5, alpha = 10, n\_estimators=40)  
  
our\_xgbooster.fit(training\_data,training\_target)  
strong\_accuracy\_test = our\_xgbooster.score(test\_data,test\_target)  
print("XGBoost Accuracy:")  
print(strong\_accuracy\_test)

Output:

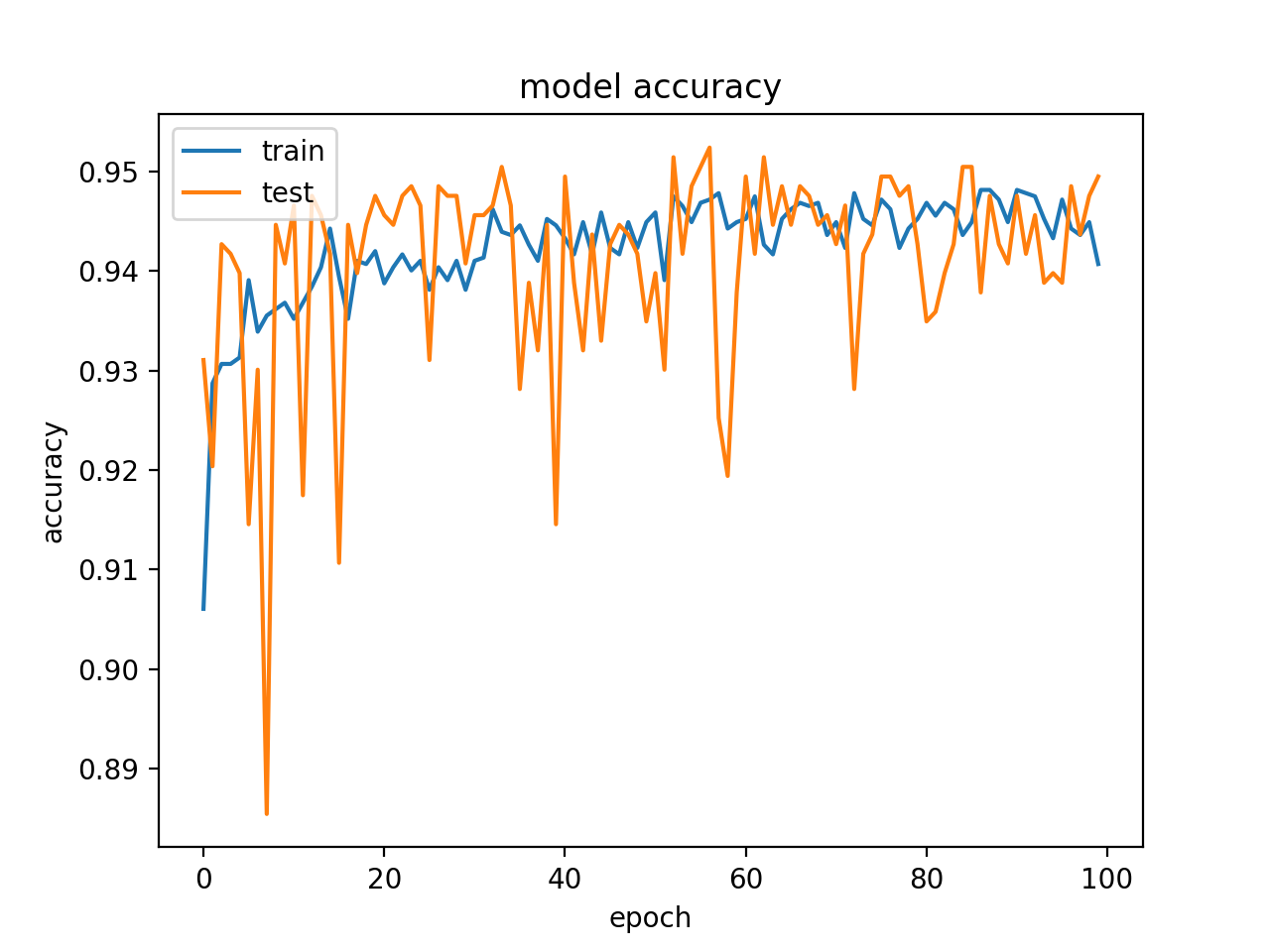


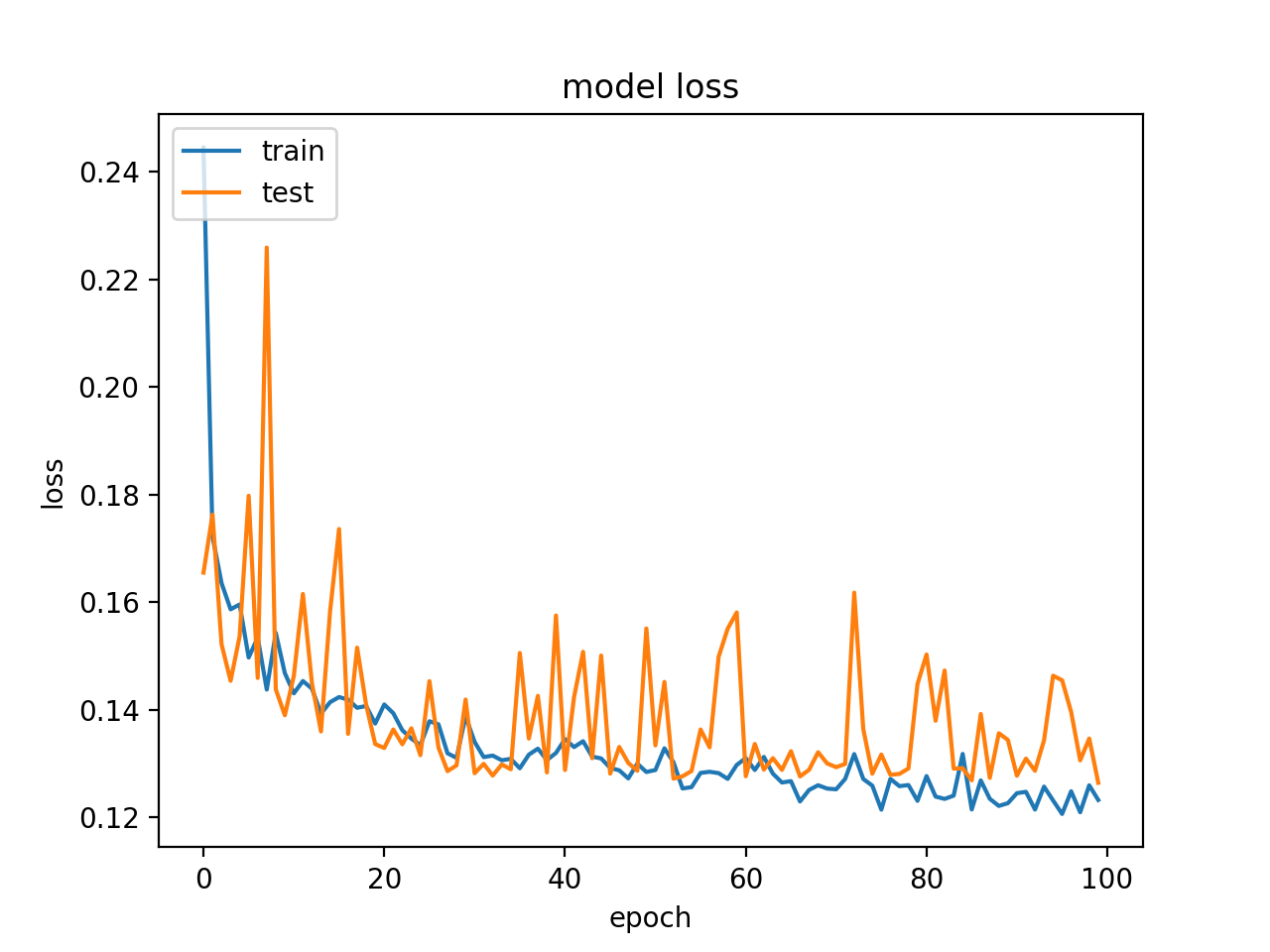
After running a single decision tree and then the XGBoost and tuning the hyperparameters, we get a accuracy of 0.9456. This puts the XGBoost behind the Logistic Regression and ahead of Random Forest in performance. Next we will finish off with a MLP Neural Network.

In our MLP, I use ReLU as activation in the first two layers and sigmoid as activation in the third. We use adam as the optimizer to have a learning rate that is maintained for each network weight and separately adapted as learning unfolds. We use 100 epochs with a batch size of 10. We choose to use a large quantity of epochs to optimize the learning by updating the weights throughout each iteration. Our final output shows us the accuracy of the model over the epoch iterations and the loss over the iterations as well.

## --------------- Keras/TensorFlow Neural Network ---------------- ##  
  
neural\_data = np.loadtxt('stats\_noheader.csv', delimiter=',')  
  
# split into input (X) and output (Y) variables  
X\_neural = neural\_data[:,0:47]  
Y\_neural = neural\_data[:,47]  
  
# split into 75% for train and 25% for test  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_neural, Y\_neural, test\_size=0.25, random\_state=7)  
  
# create model  
model = Sequential()  
model.add(Dense(12, input\_dim=47, kernel\_initializer='uniform', activation='relu'))  
model.add(Dense(8, kernel\_initializer='uniform', activation='relu'))  
model.add(Dense(1, kernel\_initializer='uniform', activation='sigmoid'))  
  
# compile model  
model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])  
  
# fit the model  
history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=100, batch\_size=10)  
# list all data in history  
print(history.history.keys())  
# summarize history for accuracy  
plt.plot(history.history['acc'])  
plt.plot(history.history['val\_acc'])  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train', 'test'], loc='upper left')  
plt.show()  
# summarize history for loss  
plt.plot(history.history['loss'])  
plt.plot(history.history['val\_loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'test'], loc='upper left')  
plt.show()

Output:





As we see from the figures above, the model started performing better over each epoch iteration. The accuracy in the best iterations was on par and even better than the Logistic Regression accuracy in the low 0.95 range. The loss decreased over time as well which is good to see as this shows us that between the training and validation data, the summation of errors made for each value became less over time.

**Executive Summary**

After running multiple predictive models to see how accurately we can predict if a player will be an NBA All Star or not, we can rank the performance of the models as follows:

1A) **Logistic Regression**

1B) **MLP Neural Network**

2) **XGBoost**

3) **Random Forest**

Logistic Regression was the simplest method to run and performed extraordinarily across the board. The MLP Neural Network using Keras and TensorFlow backend performed extremely well also and could be preferred due to the implementation of multiple iterations and the continuous updating of weights. XGBoost performed very well also, and Random Forest did too although compared to the rest, it was the worst performing model.

As far as recommendations go for this analysis, it can be noted that by using a combination of advanced metrics that are newer to basketball and more simple metrics that are easily calculated and can be just by watching the game from home, we are able to accurately predict quite well if a player will be an All Star or not. One thing I would change if running more in-depth analysis on this particular application would be to tune hyperparameters for each model more in-depth to find even more optimal model performance for each algorithm. Also, it might be more intriguing to use a wider range of dates, but a more condensed, filtered set of parameters to make it in the final dataset via cutoffs of other variables as opposed to just Games and Minutes Played. The data was definitely heavily biased towards the True Negatives as in the NBA with fifteen spots on each team’s roster and thirty teams in the league, only twenty four out of four hundred and fifty players make the All Star game every season, and this total of four hundred and fifty doesn’t take into account players who get called up from the G-League and acquisitions. Since our True Positives per year would only equal about 5.33% of the total data per season, it makes sense that there are so many True Negatives influencing the model performance. Therefore, the most important metrics to look at are the models performance on True Positives, False Positives, and False Negatives. In future research it may be interesting to take more explanatory variables such as physical attributes of players as well. Metrics such as height, weight, wingspan, vertical jump, and more could be interesting to run additional classification on as well as provide opportunity for clustering across the target variable as well. The game of basketball is changing by the minute, once fairly one dimensional, every movement on an NBA court is being tracked and monitored to see how it measures player and team performance. It will be interesting to see how the game and league continues to evolve over time.