Comparison of Predictive Models on NBA Game Outcomes
repo: github.com/ruupusensei/584_Project

Zain Farhat
zfarhat@hawk.iit.edu

Luke McDaneld
lmcdaneld@hawk.iit.edu

Elliot Tan
etan1@hawk.iit.edu

Abstract

Our goal is to predict the outcome of future NBA games based on historical data. We intend to use both player and team data to create a variety of predictive models. The end goal of this project is to compare and contrast different models to select the most accurate method for predicting future outcomes.

Many classifiers and prediction models already exist for this problem. Media outlets like ESPN and fivethirtyeight use their own predictive methodologies (Real Plus Minus and RAPTOR respectively) with varying degrees of success. A brief survey of the topic shows that the best performing models achieve near or below 70% accuracy in predicting the correct outcome of a given game [8]. Our goal is to implement a variety of models to try and replicate or surpass these results.

Because we acknowledge that many of these models already exist, our aim is to take a deeper dive into the accuracy of the models and analyze how the selection of predictors and models can improve or degrade the prediction accuracy. The end result of our project is to compile a comparative study of model performance as well as how different statistical approaches influence the accuracy of the models.

1. Introduction

As previously stated, the goal of our project is to use machine learning models to predict the win loss outcome of an NBA game before it occurs. There are different methods and models that can be developed to address such a problem, and in our work we propose three different modeling strategies to compare and contrast as to which is most effective at predicting the winner of future NBA games.

The first model considered for classification is Logistic Regression. Logistic Regression is a discriminative model that utilizes stochastic gradient descent to determine the optimal parameters for prediction by minimizing model loss. This is a supervised training method that requires labeled data to train the model. Logistic Regression applies the logistic sigmoid function to the output to squash the

Figure 1. Goal: predict who will win based on team performance

output between $[0, 1]$. This allows a binary classification to be made based on which side the interval the model output falls upon. Logistic Regression is the basis for more complex classification models like neural networks, so we considered it a good baseline upon which to compare our results for competing classification methods.

The second model is a Random Forest Classifier. A Random Forest Classifier uses a large number of classification trees that each use a random subset of the predictor variables in the tree model. A classification tree predicts qualitative responses for observations by choosing the most commonly occurring class after some observation based on training data. We are generally interested in not only the actual prediction, but also what proportion of the training observations fall into that class. Classification trees are made up of nodes and leaves (also referred to as terminal nodes). To grow a tree, the algorithm recursively splits a data set by using the Gini index/entropy, or some other measurement of node purity.

The Gini equation is defined as follows: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$ Node purity is a measurement of whether a node contains observations mostly from one class. For example, a Gini index of 0.5 where K = 2 means that a node is very impure as 50% of its observations are

from class A, while the other 50% are from class B. In contrast, a Gini index of 0.099 means that one of the classes represents 99% of the observations in the node, while the other class only represents 1%. This example would be considered a very pure node. The goal of a classification tree is to make the terminal nodes as pure as possible.

In practice, each tree uses at most $\sqrt{p}$ of p predictors. This reduces the chance of having collinear trees. The trees are then combined into a Random Forest Model. The model predicts the class of an observation or test data by taking the majority classification of all the trees in the Random Forest Model. This method was selected in order to compare a model not based on the sigmoid function. [3]

The third model is a Recurrent Neural Network. Recurrent Neural Networks are best suited for future prediction tasks because they are able to memorize and utilize previously occurring information. Recurrent Neural Networks learn different patterns in historical data, especially those that are sequential. This means those that patterns that occurred in separate time frames apart from each other, can be stored and remembered in terms of weights. The aim in implementing a Recurrent Neural Network is to examine whether or not these patterns aid a lot in predicting who will win a future NBA game as previous games and team performances are the basis of all model predictions in this experiment.

If you were to ask a person who will win an upcoming NBA game, they might jog their memory and count how many wins or losses prior to this game does each team have and based on those numbers predict a winner. Recurrent Neural Networks do the same thing, only they can withstand much more memory and also take into account other features of team performance that can help in predicting which team will win.

Our work is also influenced by other research done in this area, examples of which can be found at: [1], [4], [2], [6].

1.1. Programming Language

The Python programming language was used along with its Machine/Deep Learning frameworks scikit-learn, Tensorflow, and Keras. The Python packages Pandas and Numpy were also employed for their data processing utility.

2. Data

For this project we wanted to initially base our models on team level metrics and statistics. In order to train our models and have large test sets we wanted to obtain a historical record of as many games as possible. Despite wanting a large available sample, we also wanted to use more recent data when possible as league averages and trends have shifted over the course of the league's history.

We settled on the games data from this Kaggle repository maintained by Nathan Lauga [5]. The games.csv file found in that repository contains data for regular and preseason games from the 2003 season to the 2020 season; playoff games are excluded. For this study the 2003 calendar season was dropped as this was a year before the Charlotte Bobcats (now Hornets) entered the league leaving only 29 teams. We deemed there was still sufficient data available with the remaining seasons and saw no issue not including this season furthest in the past.

2.1. Kaggle Data Features

The following features were extracted from the raw data and will eventually makeup the training and test data: Game Date, Home Team ID, Visitor Team ID, Total Points Home, Field Goal % Home, Free Throw % Home, 3 Point % Home, Total Assists Home, Total Rebounds Home, Total Points Away, Field Goal % Away, Free Throw % Away, 3 Point % Away, Total Assists Away, Total Rebounds Away, and Home Team Wins. The final value, a binary 0 (loss) or 1 (win) indicates whether or not the home team was the winner of the game. Home team wins serves as the y value for our models, we aim to classify whether a match-up between the two teams will result in a win or loss for the home team.

2.1.1 Correlations Within the Data

Here we present the correlation matrix generated with the Pandas package in Python. The features representing positive performance by the home team all have positive correlation with the home team winning, whereas the away team features all negatively correlate with the home team winning.

Certain statistics also correlate as intuition might suggest. Points scored and field goal % have a strong correlation; the greater percentage of shots a team makes, the more they score. Home and away 3 point shooting % have no correlation at all, this makes sense as the rate at which one team makes 3 point shots should not affect the rate the other team makes them.

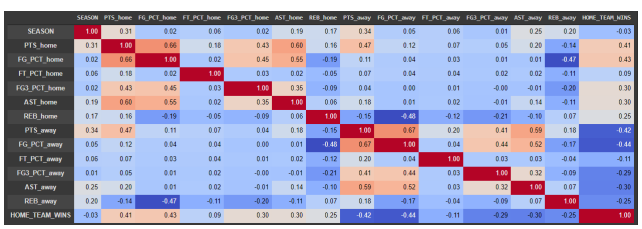

Figure 2. Correlation Matrix among Raw Features

2.2. Data Processing

In order to train the model to predict win loss outcomes based on the features in the data, processing had to be done to represent each game based on the historical performance of each team before the game. Each feature vector consists of the home and away team's performance averages over the last n games, and based on this information the model will predict a home win or loss. Beyond this, our aim was also to examine how much historical data is necessary for the model to make sound predictions. For various models we tested from 1 to 80 game historical windows, considering a sample size of just 1 previous game to nearly an entire season (the length of the regular season is 82 games).

Two main custom functions were employed to prepare the data to train the model. One function takes a team name, a date, the raw data as a Pandas dataframe, and a number of games to average over historically. The second function calls this function repeatedly as it iterates over every game entry in the raw data. The second function outputs a Numpy array where each individual event (game) is represented as the historical averages over the last n games, and the last item in each row is the binary Home Team Wins indicator [8]. Features are then normalized after the fact so that all values in the final feature vectors fall in the $[0, 1]$ range. See the associated .ipynb file for details.

2.3. Final Data sets

The resulting data consists of 23274 vectors, each of which represents a single game event. There are 12 predictive features in the feature vectors (see image below). The corresponding y vectors contain only the single binary class variable .

For logistic regression data with a historical average over 1, 5, 10, 20, 30, 40, and 50 previous games were prepared. These were then split into train and test sets using scikit-learn's train test split function. 80% of available data was used to train the Logistic Regression models, and 20% was left aside as test data.

```
Normalized averages over last 10 games for each team
PTS_HOME, FG%_HOME, FT%_HOME, 3PT%_HOME, AST_HOME, REB_HOME
PTS_AWAY, FG%_AWAY, FT%_AWAY, 3PT%_AWAY, AST_AWAY, REB_AWAY
array([0.88203547, 0.8723211 , 0.8748612 , 0.70637899, 0.76539589,
       0.78205128, 0.94050343, 0.83247423, 0.91545353, 0.53508246,
       0.75872093, 0.86545455])
```

Figure 3. Feature Vector Used in Logistic Regression and Deep Learning Approaches

3. Logistic Regression

As Logistic Regression is one of the fundamental classification models, it was our initial choices to include in this comparison study. In order to model Logistic Regression, we employed scikit-learn's Logistic Regression function. Training data was prepared as described in the previous section, and accuracy was calculated on the remaining test set representing 20% of all the games in the entire data set. Below is a figure showing how the number of historical games affects the prediction accuracy on the test set.

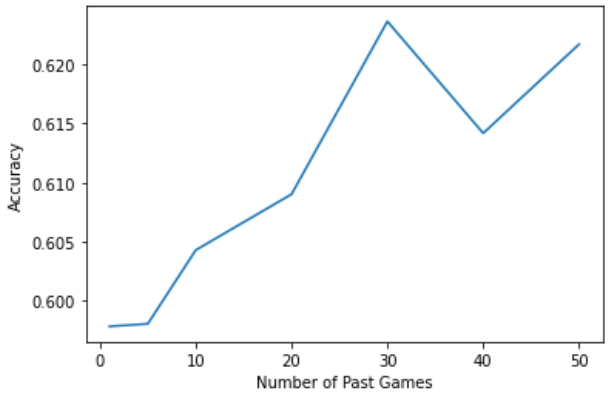

Figure 4. Accuracy of Logistic Regression Per n Game Window

Accuracy steadily rises until it peaks at 30 historical games. The increase in accuracy is actually not all that large between 1 and 30 games, but there is enough of an increase to suggest the extra information helps the model find a better classification boundary. It is also important to note that when the historical window increases the 30 game window contains all 20 games that were in the 20 game window plus 10 previous. This means that the longer window does contain the same bias as the shorter window. The results seem to suggest that more games helps establish a trend and smooth out some early outliers, but that the benefits plateau somewhere around 30 games for this approach.

4. Random Forest Classifier

The idea was to classify a game as a win or a loss for the home team. In order to do this, We took the data and made columns of the differences between the home and away team's attributes (difference between 3 point percentage, assists, rebounds etc). We then fitted a Random Forest Model over those variables for predicting the label of 1 or 0 (1 = home team wins, 0 = away team wins) over the 2004 to 2019 season.

In reality, we would not have any statistics about the games that had not happened yet, so in order to predict the outcome of a game with 2 teams (team home and team away), we used the respective team's historical averages for the features we needed. The model was tested using various values of n, where each team's attributes were aggregated over their last n games, and this was done for the entire 2020 season. The values for n used with the Random Forest Classifier are: $(5, 10, 20, 30, 40, 50, 60, 70, 80)$. After the teams were aggregated over the past n games, the differences be-

tween the home team and away team were found, and those differences were passed to the model as testing data for each observation. From this, the model would predict who would win and lose.

Since predicting the outcome of a game using the points would be trivial (since whoever scored more points won), the points scored attribute was removed.

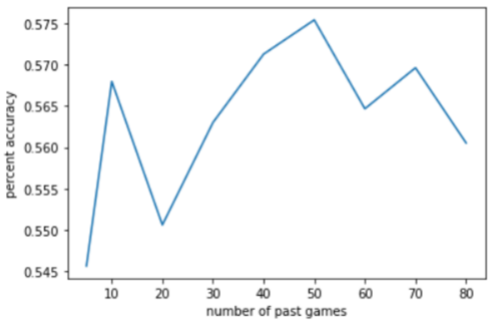

Figure 5. Accuracy of Random Forest Classifer Per n Game Window

Though the accuracy for all the 2020 season games varies only slightly, one can see that the optimal number of past games to use with the Random Forest is 50. This suggests that using too many or too few games results in a less complete estimate of the team's scores.

5. Deep Learning Approach

The third approach we took is a Deep Learning one, we sought to harness the power of Neural Networks to help in predicting the win/loss of a future NBA game. Deep Learning is better than Machine Learning in that it performs much better with large scales of data. In our data set, we have data for 16 full NBA preseason and regular season games (playoffs have been excluded), which is quite large.

5.1. Recurrent Neural Network

In order to efficiently predict the winner in a future NBA game, we must be able to retain both team's performance in past games. Based on the results obtained using logistic regression, we decided to use features from the last 30 games as that yielded optimal results. The reason why the last 30 games are most predictive can be explained as follows: in a regular season there are a lot of factors that negatively affect a team's performance such as injuries, road trip exhaustion, trying new plans and team formations, etc. To combat these negative factors, we take into account the last 30 games of an 82 NBA games season. This can capture the optimal performance of a team and reduce the number of bad nights thus being fair to both teams in predictions.

A Recurrent Neural Network (RNN) acts like a feed forward neural network but with an extra feature: it points backwards. This means that at each time step given, we have two corresponding weights, one for the current time step we are at like in any regular feed forward neural network, and another for the previous time step, the one that memorizes and detects patterns for the future. A typical Recurrent Neural Network can be thought of as a time traveller in the sense that it unrolls through time, it does not back-propagate like a regular feed forward neural network, it developed it's own concept for it called back-propagation through time (BPTT.)

5.1.1 Network Architecture

Our Recurrent Neural Network is composed of 5 Simple RNN layers, all activated with the relu function. After experimenting with the number of layers and testing with different Dense and Dropout layers, results suggested that Simple RNN layers are needed. Extensive experimentation set the number of Simple RNN layers at 5 and that is what yielded the best accuracy.

5.1.2 Loss Function

The loss function used for our Recurrent Neural Network was categorical cross entropy. Although binary cross entropy is what first comes to mind for a win loss, but the nature of our data and the use of tensors forced us to use categorical cross entropy which still serves the same purpose as binary cross entropy.

Loss function for Categorical Cross Entropy:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\ell_{y_i \in C_c}\log p_{model}[y_i \in C_c]$$

5.1.3 Optimizer

Optimizer used for the Recurrent Network trained is Stochastic Gradient Descent with a learning rate of 0.01, decay of $1 * 10^{-6}$, and momentum of 0.9. Other optimizers such as adam, rmsprop, Adadelta, and Adagrad were tested but all gave either much lower or slightly lower results.

5.1.4 Results

We used accuracy as our metric because our scenario is a simple win/loss classification prediction. Our Recurrent Neural Network was able to achieve a 91.70% test accuracy, and a test loss of 0.674.

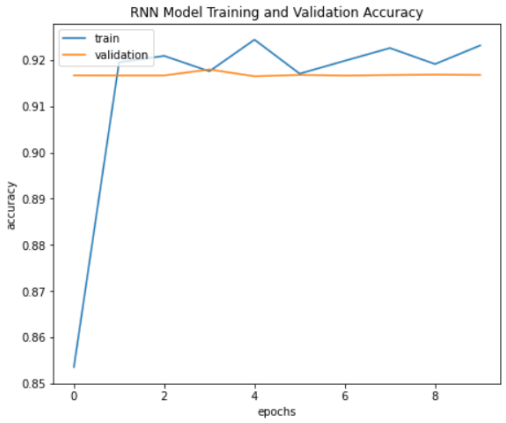

Figure 6. RNN Training vs. Validation Accuracy Curve

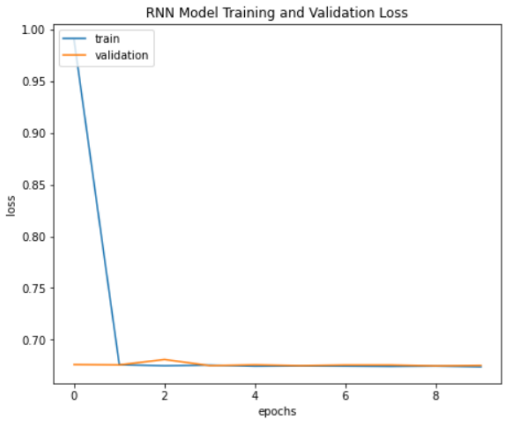

Figure 7. RNN Training vs. Validation Loss Conversion

5.2. Feed Forward Neural Network

A typical Feed Forward Neural Network (FFNN) was also trained on our data on the last 30 games as well. This was trained to compare with our Recurrent Neural Network and to understand the importance of memory in future predictions.

5.2.1 Network Architecture

For the sake of comparison, this was kept simple at one Dense layer with 12 units and relu as its activation function. Experimentation with more Dense and Dropout Layers did not increase its performance.

5.2.2 Loss Function

Unlike the Recurrent Neural Network, the loss function used for this Feed Forward Neural Network was binary cross entropy.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i))$$

5.2.3 Optimizer

The optimizer used for Feed Forward Neural Network was rmsprop. Other optimizers like Stochastic Gradient Descent, Adam, Adadelta, and Adagrad were tested but they did not perform well. With binary cross entropy as our loss function, rmsprop was able to complement it by efficiently optimizing it.

5.2.4 Results

Our Feed Forward Neural Network was able to achieve a 58.99% test accuracy, and a test loss of 0.670.

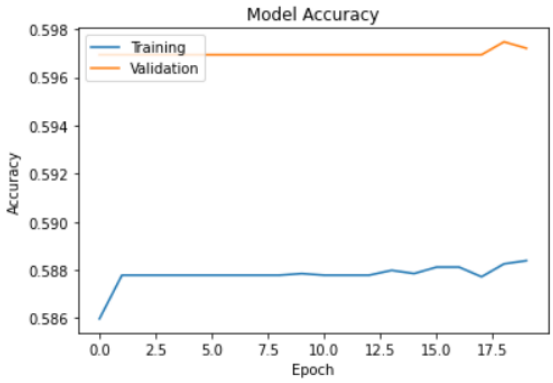

Figure 8. FFNN Training vs. Validation Accuracy Curve

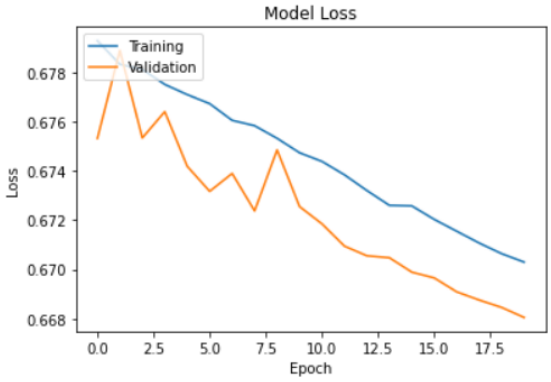

Figure 9. FFNN Training vs. Validation Loss Conversion

6. Ablation Studies

6.1. Long Short Term Memory

Instead of a Recurrent Neural Network, we wanted to test the efficiency of a Long Short Term Memory (LSTM)

model. Long Short Term Memory models can deal with Recurrent Neural Networks greatest weakness, vanishing and exploding gradients. Although Long Short Term Memory models have gained success in solving novel problems in Deep Learning, it failed in our case and was not able to compete with the Recurrent Neural Network.

6.1.1 Network Architecture

Our LSTM model was a simple one, it was composed of one LSTM layer with 6 units and one Dense layer with 256 units, both activated with the sigmoid function. Dropout layers and more Dense layers did not increase performance and only dropped it.

6.1.2 Loss Function

Binary Cross Entropy was used for the LSTM model.

6.1.3 Optimizer

Optimizer used for the LSTM Network trained is Stochastic Gradient Descent with a learning rate of 0.01, decay of $1 * 10^{-6}$, and momentum of 0.9.

6.1.4 Results

Our LSTM Network while training froze on one validation accuracy and would not improve no matter how many epochs were used. Different loss functions and optimizers were used to test this behaviour and it still persisted. With all this, test accuracy was at 59.66% and test loss at 0.676

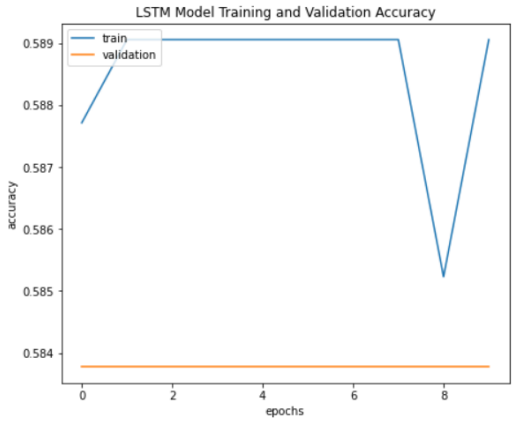

Figure 10. LSTM Training vs. Validation Accuracy Curve

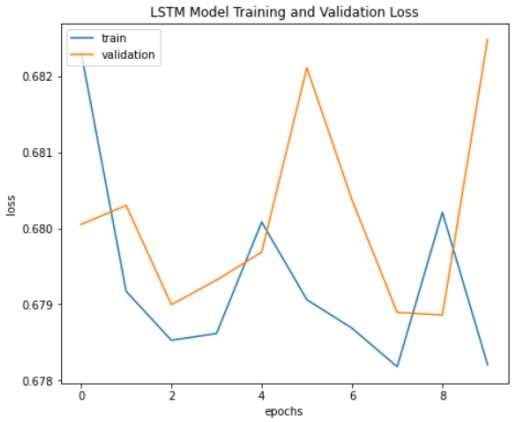

Figure 11. LSTM Training vs. Validation Loss Conversion

7. Comparison of Models Results

Model	Logistic Regression	Random Forest	FFNN	**RNN**	LSTM
Highest Test Accuracy	0.624	0.575	0.588	**0.917**	0.597
Test Loss	–	–	0.667	**0.674**	0.676
Optimal n Previous Games	30	50	30	**30**	30

As the above figure shows, the RNN was the clear standout of the three main models tested. The amount of accuracy increase seen from the other models, all of which performed between lower 50s and upper 60s, suggests that there may be conflating factors causing its performance to skyrocket. Using the team-level raw statistics as our features, we were not expecting to yield such high level results due to the somewhat primitive nature of this data. Perhaps the RNN is much better suited to the historical learning aspect associated with the historical data used; the reason for this high performance deserves greater examination.

Among the non-deep learning approaches, Logistic Regression was able to slightly outperform the Random Forest. Random Forest did forgo the use of points as a predictive metric to facilitate better learning from other features, perhaps finding a way to incorporate points into this model help it gain some accuracy.

8. Considerations for Future Implementation

8.1. Player Data

The modeling performed in this study focuses on team level metrics. The next natural step to capture more nuance game to game, as well as over different seasons as players change teams, is to include individual player performance as features. In any given season there are approximately 450 players in the NBA as each roster consists of 15 players. Roster spots are subject to change due to injury provisions as well as other factors like two-way contracts with the developmental G-League.

Including player level data in the model would likely increase its prediction capabilities. As there are only 5 players per team on the court at one time, a single player can have an outsize impact on the outcome of a game. In fact, statistical methods like wins above replacement (WAR) suggest that the best players in the NBA are worth far more than the average player. Tracking when and where these players are playing along with their recent performance will introduce more useful predictive variables into the model.

The main issue with introducing player level data is the overhead involved in the initial setup. This greatly increases the amount of data being handled and work would have to be done to streamline updating the data as games occur. Currently the models work with statistics that are relatively easy to calculate, but once the initial setup is complete, preparing and updating feature vectors with player data can be automated as well.

8.2. Feature Engineering with Advanced Metrics

Another consideration for improving the predictive power of our data is implementing some of the more established advanced metrics to measure team performance into our model. Instead of relying on raw statistical totals as our model currently does, we might be able to glean a performance increase by using metrics like offensive and defensive rating. These metrics aim to measure how many points a team scores per 100 possessions, and how many points a team allows per 100 possessions. These metrics control for the amount of possessions a team has where there is an opportunity to score and how efficient the team is either scoring or defending the basket in that situation. These metrics control for factors like points scored or allowed that can fluctuate between individual games due to factors like the number of possessions available within a game.

Advanced metrics also can be used on player level raw data as well. Metrics like PER (player efficiency rating), WAR (wins above replacement), VORP (value over replacement player) all aim to quantify the contributions of players within a single metric. Offensive and defensive ratings per 100 possessions can also be calculated for individual players in a similar manner to that of team offensive and defensive ratings.

A final consideration for feature engineering is the incorporation of Elo ratings into the model. The Elo rating system is used to measure the relative skill of different entities participating in a zero-sum game [7]. Elo ratings are updated after the result of every game to reflect the current relative ”skill” of each team in the league. Elo also biases towards quality wins and quality losses: teams with high ratings that lose to low rated teams will see a larger decrease in rating than if they lost to an equally skilled team. Elo ratings are widely used in statistical analysis of the NBA and for predictive purposes, incorporating Elo ratings into our model seems like a valuable step in feature engineering.

8.3. Predicting Wins Against the Spread

One practical application of our model would be to see if beyond predicting a simple win loss outcome we are able to train it to predict winners against the spread. While predicting wins and losses is certainly an interesting experiment, consistently predicting wins against the spread has the potential to be a money-making endeavor.

Odds-makers use their own predictive models to determine how many points a given team is favored to win by and make necessary adjustments to hedge against what side of the outcome the public is placing their wagers on. Being able to predict how often a team covers the point spread would allow for a more calculated methodology for placing wagers. For example, a starting point for this sort of modeling would be to predict how many points the model expects a certain team to win by and calculate model loss as the predicted amount versus the actual margin of victory. These predictions could then be compared to the spreads set by odds-makers to see how closely our model matches those of gambling markets.

8.4. Playoff Modeling

Our study focuses on regular season outcomes and does not take into consideration playoff or eventual championship results. In some ways playoff games lend themselves better to predictive tasks. The NBA is often considered a ”chalk” league in the playoffs, that is to say upsets are rare and the better team often wins in a best of seven games series. This would also be a good situation to test the effectiveness of individual players on game outcomes as the same players will be playing head to head in consecutive games.

Playoff modeling also has gambling applications as well. Predicting the outcome of a playoff series is a popular gambling activity. Even beyond that futures bets on who will win the NBA championship before or during the season are common as well. These tasks could be explored as extensions of the current model as it exists today.

9. Conclusion

Up to this point, we have implemented Logistic Regression, Random Forest, Feed-Forward Neural Networks, Recurrent Neural Networks, and Long-Short-Term Memory Neural Networks to predict the outcome of regular season NBA games. We prepared out data such that the feature vectors upon which these predictions will be based are determined by historical statistical averages over a window of a certain number of games. The results suggest that on average the models perform best when approximately 30 games worth of historical averages are available. A Recurrent Neural Network appears to be the best deep learning approach for this task; Logistic Regression performs second best compared to all other deep learning and classification approaches.

While there is much more to be explored in regards to this topic and potential applications of this model, we believe the work we have done up to this point is an excellent foundation upon which to base future experiments and modeling tasks.

References

[1] M. Ahmadalinezhad, M. Makrehchi, and N. Seward. Basketball lineup performance prediction using network analysis. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 519–524, 2019. 2

[2] J. A. Caliwag, M. C. R. Aragon, R. E. Castillo, and E. M. S. Colantes. Predicting basketball results using cascading algorithm. In *Proceedings of the 2018 International Conference on Information Science and System*, ICISS '18, page 64–68, New York, NY, USA, 2018. Association for Computing Machinery. 2

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning: With applications in R*. Springer, 2021. 2

[4] H. Jia, C. Ren, Y. Hu, Y. Chen, T. Lv, C. Fan, H. Tang, and J. Hao. Mastering basketball with deep reinforcement learning: An integrated curriculum training approach. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1872–1874, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. 2

[5] N. Lauga. Nba games data, Nov 2021. 2

[6] S. Li. Revisiting the correlation of basketball stats and match outcome prediction. In *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, ICMLC 2020, page 63–67, New York, NY, USA, 2020. Association for Computing Machinery. 2

[7] N. Silver and R. Fischer-Baum. How we calculate nba elo ratings, May 2015. 7

[8] J. Weiner. Predicting the outcome of nba games with machine learning, Jan 2021. 1, 3