Project Report: Predicting Three Pointer Accuracy of NBA Players

Jayesh Paunikar (jap452), Manas Koppar (mk2235), Trevor Xavier (tx28)

Introduction

Like any sport, Basketball has evolved through time. Three-pointer shooting is valued highly in modern Basketball. On paper, it's just one extra point. In technicalities, it's just 22 feet from the corners and 23.75 feet from the top of the key. The 3-pointers are becoming a more monumental formula to success in the league, year by year. The 3-pointer differential is the difference between how many 3-pointers (per game) a team makes, subtracting the amount of 3-pointers their opponent makes (per game). The advantage teams have gained for the last five seasons by becoming leaders in 3-pointer differentials is widely impressive but not surprising. Nearly 70% of the teams with a positive 3-pointer differential made it to the playoffs.

Problem

We wanted to devise a method to predict a player's three-point shooting accuracy. We also built a model to classify whether an NBA player will shoot three-pointers above a certain threshold percentage (above average at 36.5%).

Data

We began with a dataset from nbastats.com, which had data related to Game Attribute Fields, and scraped data from basketballreference.com to get the physical attributes. To fully represent all the features associated with a player, we combined these two datasets using vlook-up to form a final dataset with all the features required and used it for training the models. We used the data from 2017-2021.

Preprocessing

From the dataset, we removed the entries with missing values in the feature TOPG. We also processed string fields like WingSpan to real numbers, which is valuable for the models. Similarly, the position of each player was one hot encoded to enrich the dataset.

Our dataset has 2445 rows and 34 features, out of which we explored the above features. We did an 80-10-10 train, validate, test split. We standardized the dataset to transform the features to comparable scales. In total, we removed 105 entries that were missing important feature information as represented by the NaN in the dataset.

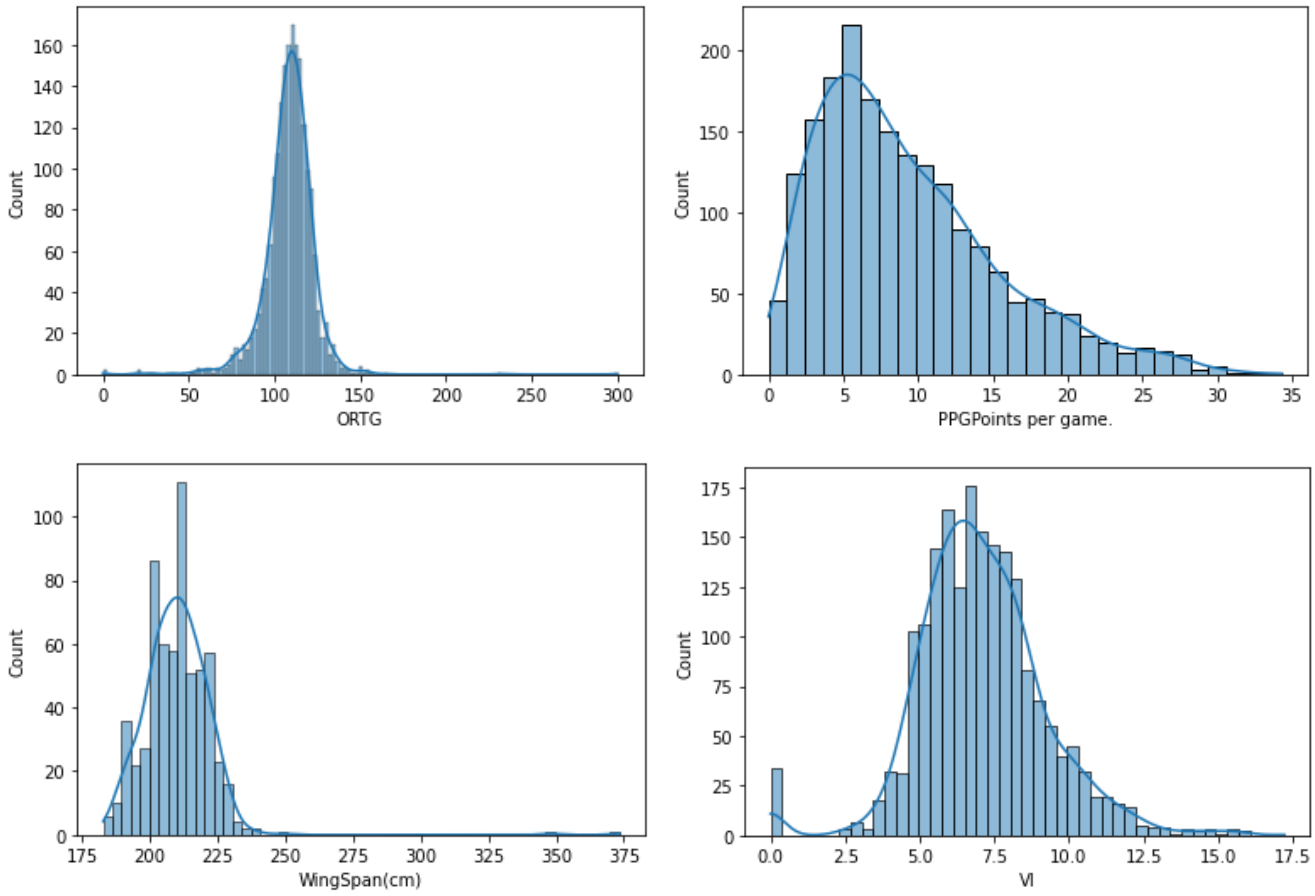
The dataset fields used for training the models:

Game Attribute Fields

Position, GP- Games Played, MPG- Minutes Per Game, USG(Usage rate)- the percentage of how long the player is handling the ball,TO(Turnover rate)- The rate at which the player loses the ball, FTA- Free throws attempted,FT%- Success percentage of free throws, 2PA- 2 pointers attempted,
2P%- Success percentage of 2 pointers, PPG- Points per game, RPG- Rebounds per game
APG- Assists per game, SPG- Steals per game, BPG- Blocks per game, VI- Versatility Index,
TOPG- Turnovers per game, ORTG- Offensive rating, DRTG- Defensive rating

Physical Attribute Fields
Height, Wingspan, Age

Descriptive Dataset Statistics



The average three-point percentage of the players during the NBA season was 30.62%. This was due to a massive outlier of 210 players who didn't play valuable minutes or shoot a three-pointer. Disregarding them, the average bumps up to 34.15%, which is still lower compared to the elite shooters who usually perform above 38%. The majority of NBA players were at the guard position as well. Finally, the ORTG(Total points achieved by the player per 100 individual possessions) was 108.58. The mean Wingspan of the players was 209.78 cm. The mean versatility index was 7.06.

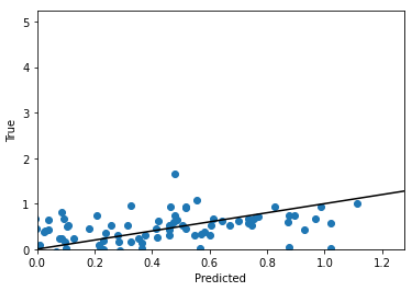
Approach

We tried to solve this problem using approaches that we learned in class. To predict a player's three-point shooting accuracy, we used various regression techniques. Namely, Linear, Ridge, Random Forest Regressor and to classify whether an NBA player will shoot three-pointers above a certain threshold percentage (above average at 36.5%) we used various classification techniques like Logistic, Ensemble Methods, Decision Tree. Finally, we tried neural networks to compare the performance against other models that we trained for regression and classification.

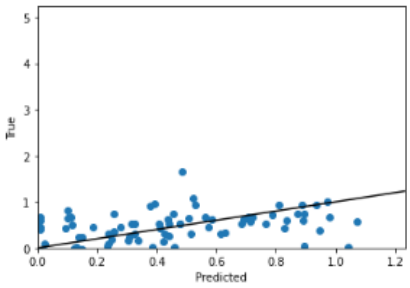
Analysis

Regression and Classification

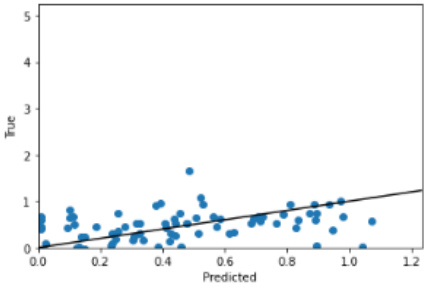
As discussed in the approach, we decided to use regression and classification to create a model that can effectively predict a player's three-point percentage and classify them as above-average to the elite three-point shooter. The average three-point percentage last year was 34%; therefore, the threshold value we as a group chose was 36.5%. Even though this increase is incremental, the overall attempts have increased by 60.1% in the last six years; thus, a slight improvement every offseason provides a massive increase in point production.
The regression techniques that we used to fit our model were linear regression and ridge regression by adding a regularization parameter. The models were fitted on the validation data set and used the mean squared error as a loss metric. Additionally, the seven different modern-day positions were one-hot encoded to increase the complexity of our position feature.



Without regularization



One Hot Encoded



With regularization

The Validation MSE error was **0.7302**, **0.7223** & **0.7223** respectively. Our model narrowed down that **VI**, the versatility index (the metric that generates the productiveness of a player), and the **Free Throw** percentage were the highest positive contributors to the three-point percentage of an NBA player. With regards to classification, the model we used to classify our data was logistic regression. We chose this model as it tends to have a low variance with the data and is much more robust with noise present in the dataset. Similarly, we found it best to use the **l1** regularizer to introduce sparsity into our parameters, effectively narrowing down the predictive parameters. The initial model of 13 features is narrowed down to one.

By implementing this model, the misclassification rate was **35.85%.** The features that this model weighs highly to increase three-point accuracy are **Free Throw** percentage and **Steals per game**.

Random Forests

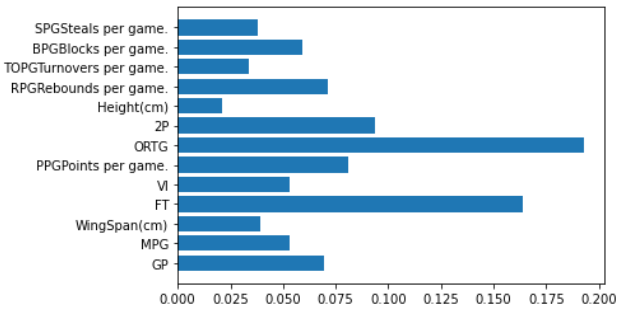
Another model we used for our classification and regression problem was the Random Forest Classifier and Regressor. They are extremely useful as they help prevent overfitting and provide a better evaluation of feature importance. Random Forests is based on using multiple decision trees to predict the mode of each input fed into the model. To prevent overfitting, we inputted multiple decision trees into the model. Therefore we used 100 decision trees on each node and implemented them into the model to predict the outcome of our training data.

For our first model which looks into predicting the three-point percentage of each player, we varied the number of trees sampled to get the best Validation MSE Error.

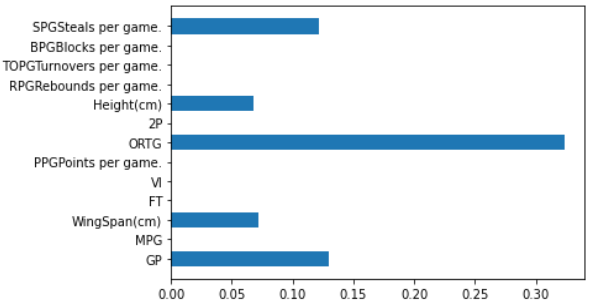
Trees	Depth	MSE
10	10	0.827
50	10	0.710
100	10	0.682

As shown in the table above, the depth is kept at 10 to make sure we have a variable amount of weak learners present, and using 100 trees gives us the best validation of a mean square error of **0.682.** Similarly, with the classification model, the number of trees used was 100 as well to stay consistent and got a misclassification error of **36.4%.**

Random Forest is also extremely useful as it helps in narrowing down the important features that help in prediction by analyzing how each input feature is weighed. Below are the two graphs that show which feature helps increase three-point percentage and be above the threshold value of 36.5%



Regression



Classification

As seen above, the regression model used free throw and an **offensive rating** of the player to be the most influential in a player's three-point percentage while the classification model indicates that the offensive rating is by far the most influential. Offensive rating is the player's ability to generate points per 100 possessions. Furthermore, to push the misclassification lower, we used the bagging classifier or ensemble to improve our classification model. Bagging is an ensemble algorithm that fits multiple models' various subsets of the training dataset and combines all the predictions from all models. Random Forest is an extension of random forests. The number of bagged models used was 150 and the samples of each subset were 30 to reduce variance and bias in the prediction. The misclassification rate when employing bagging was **35.18%**

Neural Networks

Why have we used Neural Networks?

Most state-of-the-art techniques in many machine learning tasks involving regression tasks like stock price prediction, classification tasks like object identification and classification in an image or live stream (Computer Vision using FasterRCNN, etc.), natural language processing tasks like machine translation revolve around neural networks. Neural Networks lay the base for different specialized architecture (Loops vs. No loops) and specialized nodes (Long Short Term Memory nodes, Convolutional nodes, etc.) used to train computers to learn the task successfully. These architectures and nodes are designed for neural nets to perform better at specific tasks. With enough data, neural nets have been statistically found to perform better than most traditional learning models. We opted for neural nets to compare performance against other models that we trained like linear regression, ridge regression, and random forest regression.

Type of neural network used

Since the task at hand was regression and classification and was not a time series type of data, we chose to use the basic form that involves a forward pass, error calculation, and error backpropagation to tune the weight matrices without loops.

What does our neural net look like?

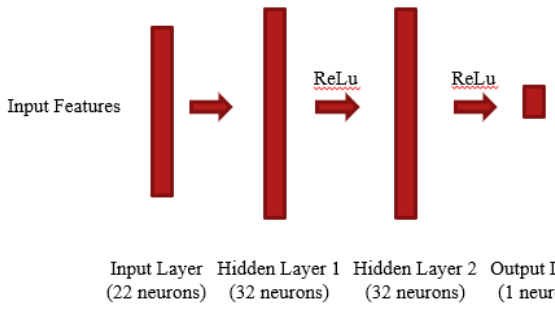


Fig 1.a Feedforward Neural Network for Regression

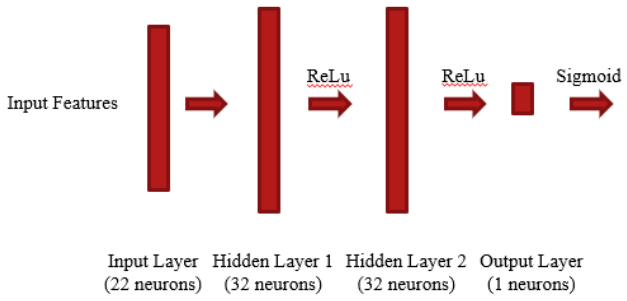


Fig1.b Feedforward Neural Network for Classification

As shown in Fig 1, it has an input layer, two hidden layers, and an output layer. The input layer takes 22 features as input. This input vector, x, is passed to the first hidden layer. The first hidden layer comprises 32 neurons where x is multiplied with the weight matrix, U, to get Ux, followed by passing it through an activation function, in our case, ReLu (let σ represent ReLu), which adds non-linearity in the network. The output of the first hidden layer is then passed to the second hidden layer, which also comprises 32 neurons. At the second hidden layer, σ(Ux) is multiplied with another weight matrix V to obtain V * σ(Ux). This is then passed through ReLu again, σ(V * σ(Ux)). This is the input to the last layer in the network, the output layer. This layer has only one node. At this node, σ(V * σ(Ux)) is again multiplied with a weight matrix W to obtain W * σ(V * σ(Ux)).

In case of regression task, W * σ(V * σ(Ux)) is the output of the network as we need to generate a continuous value, whereas, for classification, we apply the sigmoid function to W * σ(V * σ(Ux)) to get a probability distribution.
Note: Each layer is fully connected with the next layer, i.e., each neuron from a layer is connected to every neuron in the next layer. (Previously, drop-outs were used which disregarded a portion of these connections to prevent overfitting, but in our case, we don't have any drop-outs)

Choice of the size and number of hidden layers-
Since we have 21 input features, there will be in total (32*21) + (32*32) + (1*32) = 1728 ~ our training set size. That is why we could not add any more hidden layers or increase the size of each hidden layer.

Training of the neural network

We used the same training set which was used for other models. For tuning the weight matrices to our dataset, we set the learning rate to 0.000001 and ran it for 25000 epochs (but we used checkpointing and saved the models from using the best weight parameters) using Adams optimizer. We tried other optimizers like SGD and other activation functions like tanh, sigmoid but Adams optimizer and ReLu gave us the best results.

Testing

Using the neural networks for regression on the validation set, we got an MSE of 0.638 (best error among the models)
Here is how it compared with other models-

Model	**MSE**
Linear regression w/o position and regularizer	0.730
Linear regression w/ position w/o regularizer	0.722
Linear regression w/ position and regularizer	0.722
Random forest regressor	0.682
Neural Net	0.638

Using the neural networks for classification on the validation set, we got a misclassification rate of 35.14% (best error among the models)
Here is how it compared with other models-

Model	**Misclassification**
Logistic Regression	35.85%
Decision Tree	36.41%
Ensemble	35.18%
Neural Net	35.14%

Based on the performance on the validation set, the best model we had was Neural Net for both tasks.
NN on test set had the following results-
Regression Test MSE- **1.28**
Classification Test misclassification rate- **37.65%**

Robustness of the best model

To confirm the model's performance, we tried to predict on a dataset consisting of outliers, i.e., NBA players who have the best 3P accuracy of all time (excluding the players the model has seen during the training). Players such as Steph Curry, Ray Allen, and Reggie Miller are generational marksmen who would be outliers. Therefore testing our prediction based on our neural net model will help quantify the robustness of our model. The results assured that the model generalizes well as the test MSE for the regression task was **0.97**, which was better than the error of **1.28** on the test set.

Conclusion

In conclusion, the neural net has the best validation mean square error of **0.638** resulting in a test error of 1.28. Similarly, it had a model best misclassification rate of **35.14%** which led to a test misclassification rate of 37.65%, making the neural net model 62.35% accurate. We believe the model is moderately precise. All the models used have similar error and misclassification rates which indicate that the models are working as well as possible. Additionally, as displayed by our models, the two most influential factors that affect a player's three-point percentage are the Free Throw percentage and the offensive rating of a player.

Regarding the fairness of our model, we did remove players who had extremely high or low three-point percentages due to taking a lower amount of shots. For example, we took out players who attempted less than 40 three-pointers attempted so we could get a better idea of the distribution of three-point accuracy. However, we are aware that good players on bad teams might have lower shooting splits due to the higher offensive load. Therefore more analysis needs to be done on team data to prevent this and help make our model fair.

Even though we employed neural networks, we would need more data samples to perfect the model. Similarly, the three-point shot is a highly volatile metric. NBA organizations effectively make plays and strategies to maximize space production on the court. Whether it is vertical spacing around the rim or on the court spacing, team and player percentages are also reliant on them. Additionally, more data on the quarter splits of players would help too. As teams near the fourth quarter, the points generated are much more meaningful and don't supplement 'empty stats'. Therefore more data into quarters and team dynamics such as points generated through organization-centric strategies would help supplement and bolster our neural net model. Then we would be more comfortable with NBA teams using our model to evaluate talent in the league during the offseason to approach players for free agency or trading.

Bibliography

https://www.basketball-reference.com/

https://www.nbastuffer.com/

https://www.nbastuffer.com/

https://www.tensorflow.org/api_docs/python/tf/keras