ML Paper

# Abstract

The NBA has changed drastically over the past decades, with a heavy emphasis on shooting nowadays. What really separates regular NBA players from superstars is not only the ability to shoot the ball well, but the ability to know when to shoot the ball, and what a “good shot” is.

With the help of the NBA shot log dataset we found online, which recorded every shot in the 2014-15 NBA season, we were able to train a few machine learning models which would predict whether a shot will be made or not based on data such as shot distance, time left on the shot clock, and distance of the closest defender.

Most of our models were able to predict with 60% accuracy, however with the help of hyperparameter tuning, we were able to increase the accuracy of one of the models to more than 86%. This heavily emphasizes the importance of choosing the correct hyperparameters for a classification model.

# Introduction

Shot selection is as important of a skill as making the shot is. Analyzing data and help figuring out what shots are good could be helpful for both players and NBA coaches to significantly increase their chances of winning.

The NBA shot log dataset online has record of every shot attempted during the 2014-15 NBA season. Given the vast amount of data such a dataset provides, it was only natural to try and test out a few machine learning models and see if we could predict whether a shot would be a make or miss with good accuracy.

As this is a binary classification problem, we thought of using Logistic Regression as our first model. We also used a fast forward neural network, and gradient boosted trees to try and see if they could perform any better.

# Problem statement

Our goal is to train various classification models on a dataset containing information about shot distance, closest defender distance, time on shot clock, dribbles taken before the shot, and more, and predict whether the shot was made or missed.

# Technical Approach

### Data Cleaning

The original dataset came with a total of 21 columns, out of which a few of them were irrelevant. Since our objective was to figure out whether a shot attempted was “good” or not, columns such as game\_id, player\_id, and the matchup were of no use, and so removed them from the dataset. We also thought that since we wanted a generic idea of whether a shot taken was good or not, the specific player who took it and the closest defender did not matter.

That being said, we recognized that the skill of the person who took the shot could impact the outcome of the shot. To account for that, we needed a metric of how good a player was, and the most generic metric were video game ratings of the NBA. To get those ratings, we used the beautiful soup library to web scrape the rating for each player in the dataset and added it to the dataset. While running our code, we came to the realization that the dataset had some misspelled player names and had to correct that ourselves.

Finally, we had to convert the dataset into data that we could feed to our models. This means, we had to convert all inputs into numbers. We used maps and lambdas to convert columns such as “win” or “location” to map the values to 0s and 1s. We also had some missing data which we had to deal with, mostly in the shot clock column, for which we had to manually loop through the entries and fill that up with the game clock data (because under 24 seconds of playing time remaining, the shot clock turns off).

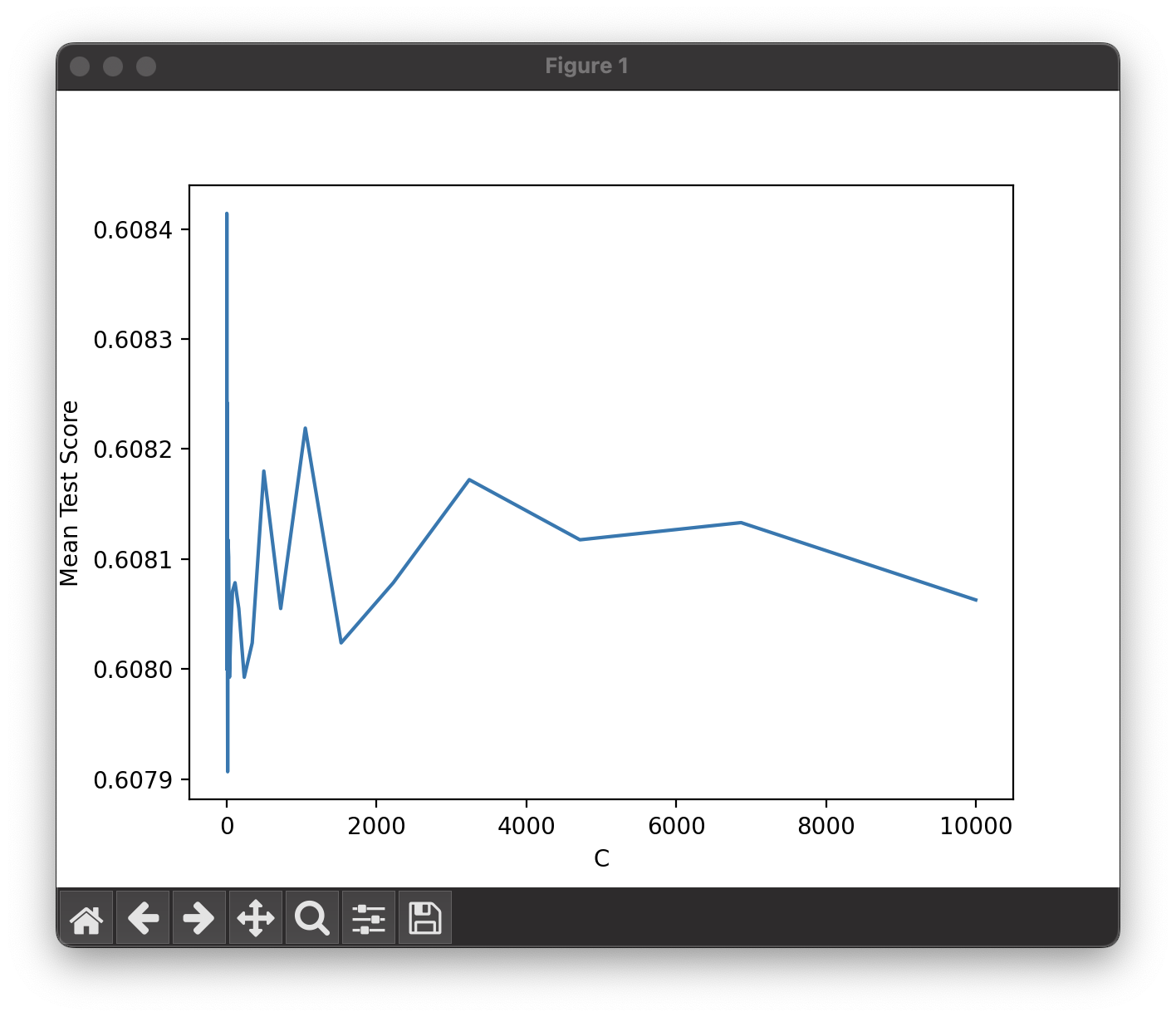
After cleaning all of this data, we were able to convert the feature vectors and target vectors to the relevant input types our models’ libraries would take to then train the models. The three types of classification algorithms we used were logistic regression, deep neural networks, and

gradient boosted trees.

### Logistic Regression

One of the simplest types of binary classification models, logistic regression takes in a feature vector, and applies a transformation to it to then get the probability for an event to happen. Depending on whether it is more or less than 0.5, it tells us whether the event is likely to occur or not.

The accuracy of logistic regression without any hyperparameter tuning was 60-61%. After trying to use cross validation using GridSearchCV in sklearn to tune the hyperparameters, we concluded that maybe the dataset was too complex for a simple logistic regression model to classify accurately. Below is a graph of the hyperparameter ‘c’ vs accuracy of the graph over a 5 fold cross validation. ‘C’ is used to prevent overfitting, however since the model was not able to fit the data at all, maybe that’s why the value of C did not matter too much.



### Deep Neural Network

For the second kind of classifier, we used a fast forward neural network. We used an architecture which comprised of 5 layers, each with 256 neurons using the ReLU activation function. Finally, on the output layer, it used the sigmoid activation, which is exactly what logistic regression uses for its model.

Since we implemented a fast forward neural network, it can self-learn the weights and basis function necessary to separate the data well and then classify it. However, after training the neural network on multiple epochs and using various hyperparameters, we were unable to do any better than logistic regression, getting a similar 61-62% accuracy in the end. Below is a graph to display the training progress, which basically shows the number of epochs vs training accuracy.

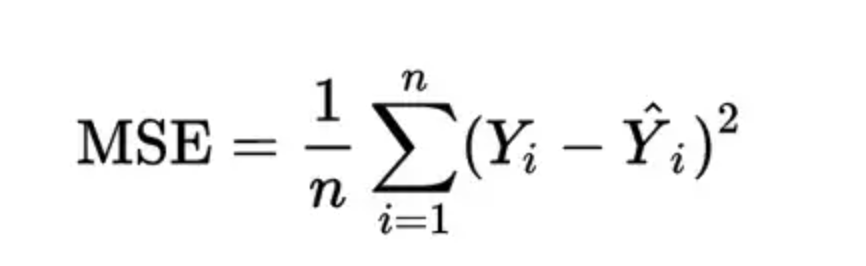
After getting 60% with both neural networks and logistic regression, we realized our model was only slightly better than a coin flip, which is not ideal. Since using a feed forward neural network also failed, we realized maybe the data was too complex to be separated at all, even using basis functions.

To solve this issue, we then looked at regression trees and gradient boosted trees.

### Regression Trees

Regression trees are decision trees that are used for the task of regression to be able to give continuous value outputs instead of discrete outputs. Regression trees use something called the Mean Square Error in a dataset to be able give a continuous value output in the function.

The Mean Square Error of n points is the sum mean of the sum of the square of the L-2 norm of all the n points, given by the formula



The goal of the algorithm used by a regression tree is to find points in the data to split and minimize the sum

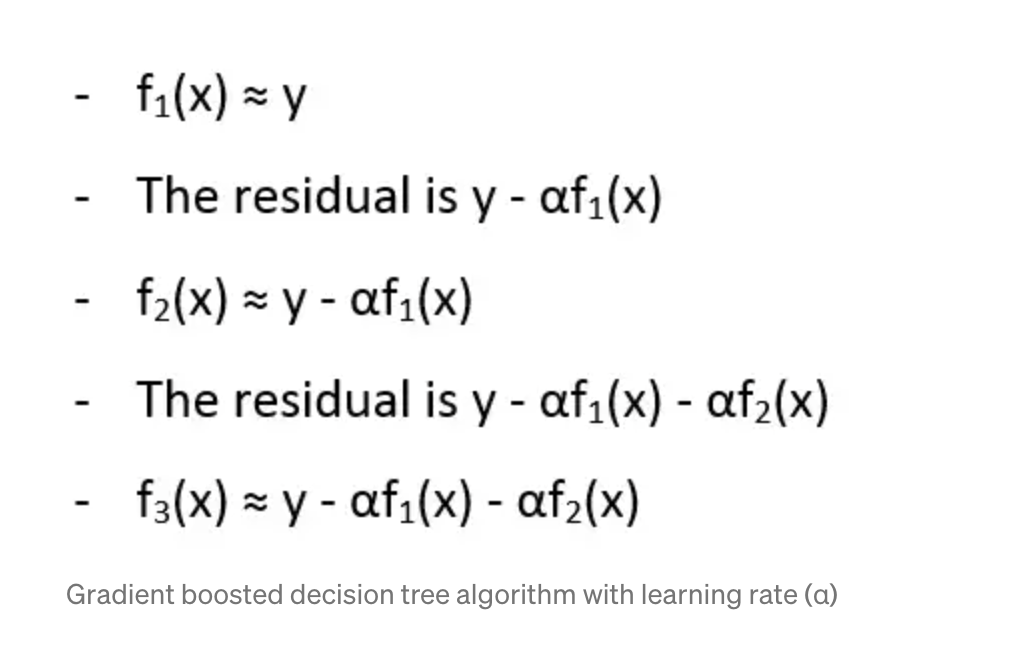
NEED MORE EXPLANATION

### GRADIENT BOOSTED TREES

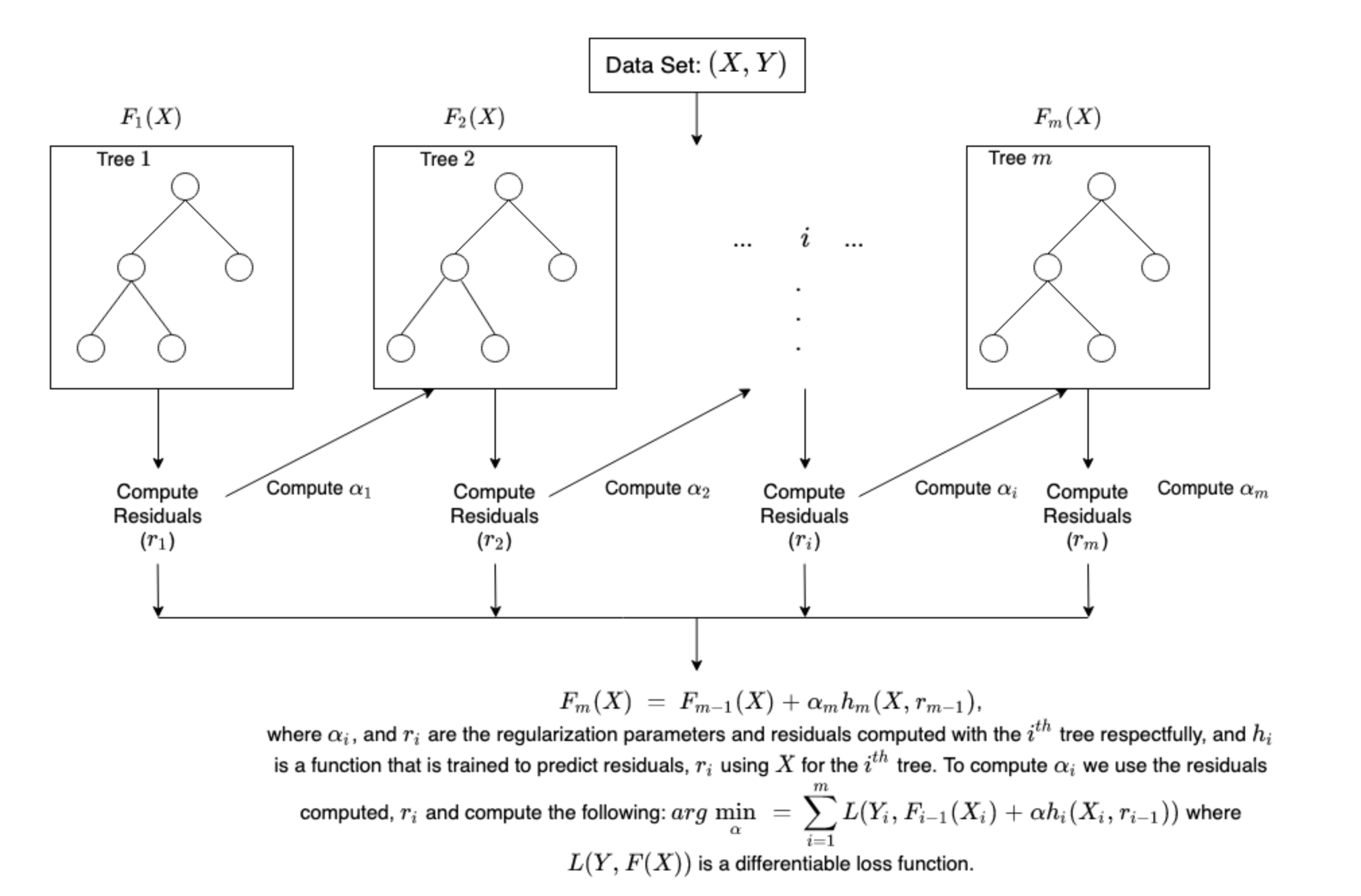
Gradient boosted trees are a combination of multiple individual trees that have been combined using boosting. Boosting, in simple terms, is a learning algorithm that sequentially learns from “weak learners” to come up with a strong learner. In the case of gradient boosted trees, the weak learners are the individual trees.

Each tree tries to minimize the error of the previous tree by focusing on the errors made by the previous tree. There are two main parameters in a gradient boosting tree algorithm – learning rate and number of trees.

The learning rate determines how quickly the model learns. A low learning rate makes the learning process of the model slow, however yields a more robust and accurate model. The number of trees used in a model normally depends on how complex the dataset is, however using too many could result in overfitting.



Briefly explain above image.



Explain this image and residuals a bit

### Gradient Boosted Trees using XGBoost

XGBoost is an efficient and scalable implementation of gradient boosted trees. To test whether decision trees could help accurately predict outcomes, we used the XGBoost implementation of gradient boosted trees. Since we wanted to prevent overfitting, we only used 1 tree in the model. But because of the scalability, efficiency, and increased accuracy, we decided to use the XGBoost implementation instead of sklearn.

Without any hyperparameter tuning, the model was already able to do noticeably better than logistic regression or the neural network, with an accuracy of over 67%. Some of the hyperparameters we looked at were the maximum depth of the tree, the minimum child weight, and the learning rate. After tuning these hyperparameters, we found that the learning rate was important to come up with a good model. With a low learning rate, we were able to come up with a model which consistently gave 86-90% accuracy on unseen data. Also, to prevent overfitting, we made sure to grid search with a low maximum tree depth and ended up setting it to 3. Below are a couple of graphs to show exactly how important the learning rate was to come up with a model with significant increase in accuracy.

**INCLUDE CONCLUSION AND RELEVANCE AND LIMITATIONS**