Predicting NBA shot success

Dhruv Chauhan, Kevin Chen

Abstract

The NBA has changed drastically over the past decades, with a heavy emphasis on strong shooting nowadays. However, the skill that has always separated regular NBA players from superstars is not only the ability to shoot the ball well, but also the ability to know when to shoot the ball.

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 through careful hyperparameter tuning, we were able to increase the accuracy of our Gradient Boosted Trees to more than 67%. The result emphasizes the importance of choosing the correct hyperparameters when training a classification model.

Introduction

Shot selection is a skill in basketball which represents a player's ability to find and go for positions with a high percentage of making an attempt at the basket. This is largely an intuitive skill, and for most people at home watching a professional basketball game who lack the necessary experience, it can be difficult to see why one player is able to make a long-range three pointer while another player fails to make a a similar shot from a shorter range.

The NBA publishes an incredible amount of data describing the 1000+ games played in the yearly season. This data includes information down to the individual shot with categories such as the attacker with the ball, any defenders, the distance between them, time left on the shot clock, and even minute details like number of dribbles (bounces) before shooting. While the primary motivation for this project was to exercise our familiarity with various machine learning models, a successful outcome for this project through analyzing the data to find out what shots are considered "good" could be helpful for professional players and NBA coaches to refine their tactics, potentially significantly increase their chances of winning. In addition, a successful classifier would be helpful for the general public in increasing familiarity with optimal basketball tactics, and potentially for human and automated sports betters, giving possibly a shot-by-shot performance evaluation for them to consider and base decisions on.

The NBA shot log dataset on Kaggle has a 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 feed-forward neural networks and gradient boosted trees to try and see if they could perform any better.

Related work

The body of work on predicting shot results is quite shallow, with an overall prediction accuracy floating in the 60-70% range. One paper approached predicting shot success using image data of the shot, with a 61% accuracy using a convolutional neural network. Our data is composed of numerical descriptions of shots being contested by defenders, so the cases are different. However, it gives us the idea of using neural networks to learn features of a "good" shot that can help to predict the outcome.

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. Ideally, an accuracy above 70% would mark an improvement over existing approaches.

Technical Approach

Dataset and 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 single shot attempted was "good" or not, columns representing metadata of the match such as game_id, player_id, and the teams involved were of no use, and were removed from the dataset.

In addition, 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. However, 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 numerical metric of how good a player was, and the most generic metric were overall ratings produced by the NBA for the video game series NBA2k. These ratings are essentially universal to NBA players, so it was a satisfactory metric to supply for our models. To obtain these ratings, we used the beautiful soup library to web scrape the rating for each player in the dataset by name and added it to the dataset.
In the process, we came to the realization that the dataset had some misspelled player names which hindered the player rating mapping. The names were manually rectified.

Finally, we had to convert the dataset into data that we could feed to our models. For example, we had to convert all inputs into numbers such as the shot clock and the game clock representing the overall progression of the match. 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).

Following the data cleaning process, the feature vectors and target vectors were converted to the appropriate input formats for use with the chosen classification algorithms: logistic regression, support vector machines, deep neural networks, and gradient boosted trees.

MODELS

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. This matches our situation as a shot has only two outcomes - success or failure. The output of logistic regression given an input vector is a probability of belonging to either class. Our model predicted a number in the range [0, 1], which we then mapped to an outcome of either 0 or 1. This was used to compare against the true label. Implemented using ski-kit learn.

SVM

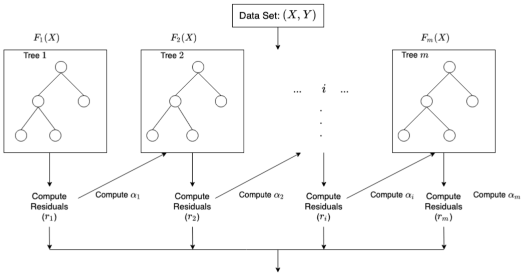
A Support Vector Machine similarly can be used to classify data into multiple distinct classes by finding a hyperplane that can separate the data. It maximizes the distance between the boundary and the nearest data point to give it the best margin to predict new inputs. Using different basis kernels, it can also be used to separate linearly inseparable data. Some of the kernels we tried included linear, polynomial, and the radial basis function. Implemented using ski-kit learn.

Deep Neural Network

For the third kind of classifier, we create a feed-forward neural network. We tried multiple architectures between 1 and 5 hidden layers with each layer having from 64 to 256 nodes. The best model based on accuracy was an architecture which was comprised of 5 layers, each with 256 neurons using the ReLU activation function. Finally, on the output layer, it used the sigmoid activation function, similar to what logistic regression uses for its final prediction. Implemented using PyTorch.

Regression Trees and Gradient Boosted Trees

The overall idea behind regression trees is to divide the feature space (i.e., the set of possible values for the input variables) into smaller and smaller regions, with the goal of creating a model that can make accurate predictions for the target variable within each region. This is done by identifying the feature and the corresponding value that best splits the data into two subsets. The process is then repeated on each subset until a certain stopping criterion is met. The final result is a tree-like model where predictions can be made by passing the input through the root node until it reaches a leaf containing the class. Since we have discreet classes and well defined data feature, we thought it would be worth trying a variation on the regression tree model to classify the data.



$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1})$,
where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

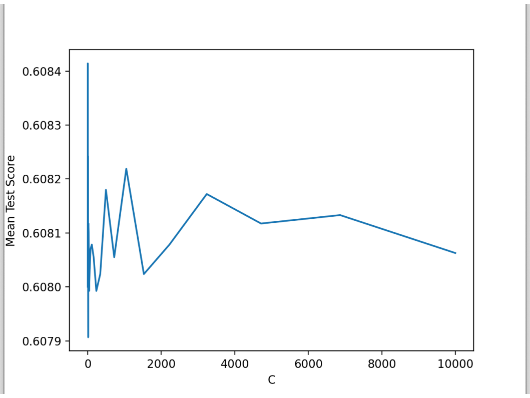
Gradient boosted trees are an ensemble learning method made from a combination of multiple individual trees. Boosting is a learning algorithm that sequentially learns from "weak learners" to come up with a strong learner. In the case of gradient boosted trees, the weak learners are the individual trees. 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. These were both hyper parameters to tune when training a model.

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.

Results

Logistic Regression

The accuracy of logistic regression without any hyperparameter tuning was 60-61%. After performing cross validation using GridSearchCV in sklearn to tune the hyperparameters the accuracy did not improve further, and we concluded that 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, the value of C did not matter much in the end.



Support Vector Machines

With our large dataset, training the support vector machine to separate the data took an exceedingly long time with both the polynomial and radial basis functions. However, no matter what we tried in the end SVM gave a similar accuracy to logistic regression at around 61%.

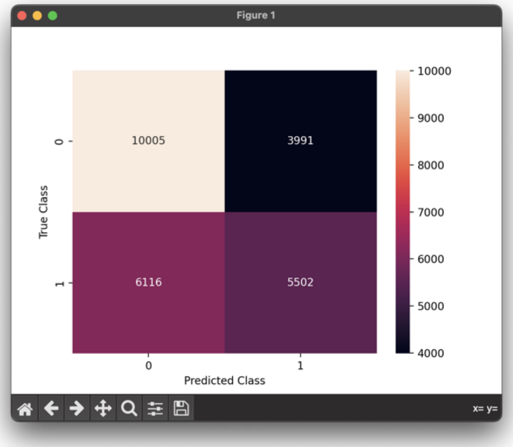


Figure 1: Confusion matrix for Logistic Regression

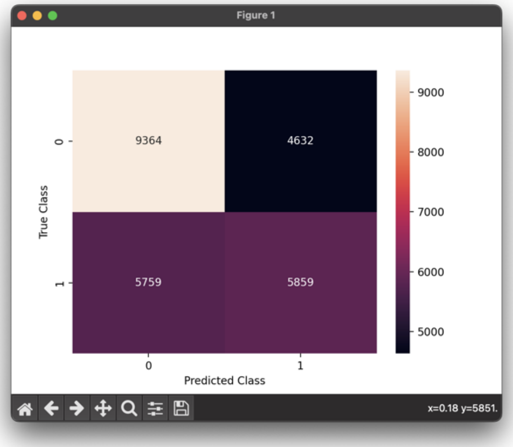


Figure 2: Confusion matrix for SVM

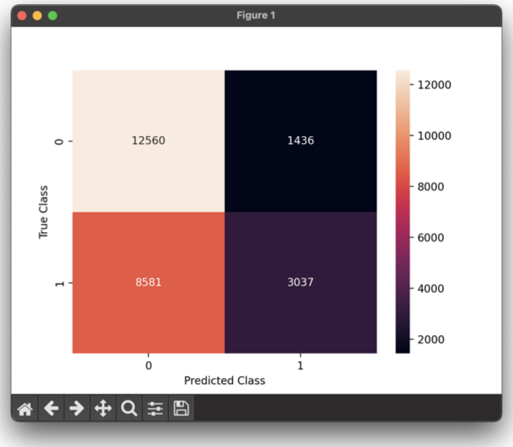


Figure 3: Confusion matrix for XGB

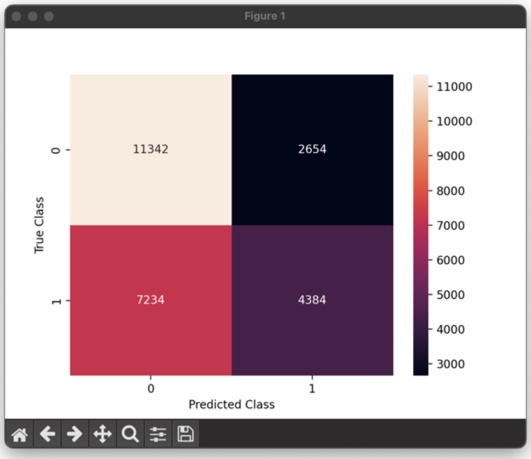


Figure 4: Confusion matrix for DNN

Deep Neural Network (DNN)

With the DNN, 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, showing 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.

Gradient Boosted Trees

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 and we used a small rate of 0.01. Also, to prevent overfitting, we used grid search with a low maximum tree depth and ended up setting it to 3. We were also able to obtain a graph of the relative importance of the features in the dataset, where closest defender and shot distance were the two most relevant.

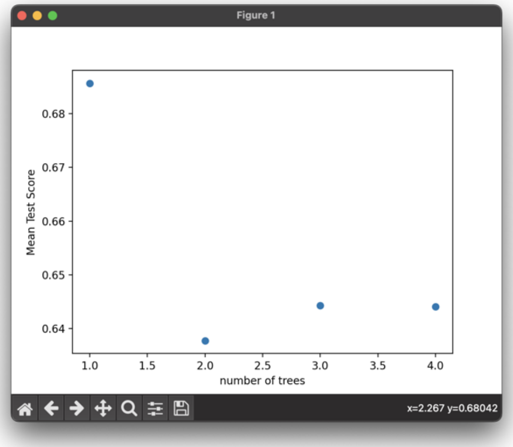


Figure 5: How the number of trees in the model affects the accuracy

Conclusion and Future Work

This project aimed to use various models to predict the success or failure of shots in the NBA based on features such as shot distance, time on the shot clock, and closest defender distance. Using the NBA shot log dataset from the 2014-15 season, four classification models were trained and tested: logistic regression, deep neural networks, support vector machine, and gradient boosted trees. The results showed that all models were able to predict the outcome of shots with only accuracy, with the gradient boosted trees model achieving the highest accuracy of over 67% after hyperparameter tuning. These findings demonstrate the potential for machine learning to provide valuable insights into shot selection in the NBA, and highlight the importance of careful model selection and hyperparameter tuning in achieving optimal performance. In addition, it seems that the distance to the closest defender and the distance from the basket the attacker is shooting from are the two most important factors in the given dataset in prediction shot success.

This project focused the evaluation of individual shot selections. In reality, the game of basketball is complex with many attackers and defenders on the court looking for avenues to score. To generalize the findings to a multiple player situation, future work could shift towards evaluating the overall position or scoring potential for a player in a particular position, and with this model it would be possible to train an artificial basketball player tutorial, helping players find the optimal spot on the court.

Limitations

The gradient boosted trees model gives an accuracy of about 68%, which is pretty good considering the dataset we have. The modeling and analysis of this dataset is subject to many limitations.

The first is that we have not separated better shooters from worse shooters, and have not considered the defender. Even though we have given a rating to every player, certain players are better shooters than others which is not always reflected by the overall rating. This might mean that one player shooting from a certain distance might have a better chance of making a shot than another, which is something that is currently missing. Similarly, we have not considered the defender guarding the shooter. Certain players are much better defenders, making it harder to make the shoot over them.

Secondly, we have not differentiated between different zones (or areas on the court), which might impact the chances of the shot going in. Even though we have the distance from the basket, the player can be in a lot of places on the court at the same distance (in a semi-circle from the basket, to be precise). On average, shots taken from the corners are more likely to go in than from the wings (45 degree angle to the basket).

Finally, since we do not have image data, we do not know the power, trajectory, and release height of the ball, all of which affect the chances of the shot going in. Despite these limitations, our best model is able to predict 7 out of 10 shots correctly, and hopefully with the help of additional data, the accuracy can be increased.

References

https://www.kaggle.com/datasets/dansbecker/nba-shot-logs

Skinner, B. (2012). The problem of shot selection in basketball. *PloS one*, *7*(1), e30776.

https://www.usab.com/youth/news/2012/07/7keys-to-good-shot-selection.aspx