1. Introduction

In this paper, we study and analyse basketball data for the NBA season 2014-15. This paper debunks various theories like the hot-hand hypothesis, where a player shot accuracy increases if he has made many successful attempts in the same game. This paper also sheds light on the clutch gene hypothesis which addresses the myth that a player performs significantly better than others in end of game situations in terms of efficiency and the points scored. This paper also proposes multiple predictive modelling techniques to predict if a shot would be ‘made’ or ‘missed’ given a certain set of parameters. To achieve this, we use a combination of two datasets, one of which includes every single shot log in the season with features like the time the shot was made, result of the shot, player who made shot, the closest defender, the distance from closest defender, the shot number and other game specific details. The other dataset depicts player specific information including the height, weight, age and other details for all players playing in the same season. A complete dataset was obtained by an inner join on these datasets. Using this complete dataset, we build a logistic regression model and a neural network model in keras, assessing the usage of the features involved in each model and discussing their results. We also discuss a recommendation model using latent factor algorithm to suggest the best defender for a particular player making the shot. As part of our factual test of the hot-hand hypothesis, we find proof that the myth is in fact true and we visualise our findings to interpret the result in a coherent manner. Overall, this paper provides a comprehensive analysis into basketball data for better insights and understanding of the game.

1. Exploratory Analysis
2. Predictive Modelling

Predictive Modelling using Deep Learning

One of the hypothesis we check if a player makes/misses the shot given set of certain parameters hold true. We use keras in python to build a sequential predictive model using a dense neural network. Neural networks generally perform better when the real-valued input and output variables are to be scaled to a sensible range. For this problem, each of the input variables and the target variable have a Gaussian distribution; therefore, standardizing the data in this case is desirable. We use selected features in the complete shot logs dataset to standardise the data. Numeric features like FGM, PTS, player\_height, defender\_height, Height\_diff, player\_weight, defender\_weight, shot\_distance and closest\_def\_distance are fit and transformed using the MinMaxScaler() function in python. The location feature is converted to a categorical binary feature with 1 being if the location is A and 0 otherwise. Other features like Player\_Age, Defender\_Age, Shot\_Number and PERIOD are converted to one hot encoding using the to\_categorical() function in keras. This is done since the values in these vectors have a smaller range and the model performs better with one hot encoded values.

A small Multilayer Perceptron (MLP) model is defined to address this problem and provide the basis for exploring different loss functions. The model expects the number of columns in the training set after one hot encoding and scaling is applied to it. The model has one hidden layer with 16 nodes and uses the rectified linear activation function (ReLU). The output layer has 1 node, given the one real-value to be predicted.

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. In order to use stochastic gradient descent with [backpropagation of errors](https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/) to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. The function must also provide more sensitivity to the activation sum input and avoid easy saturation. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less. We can describe this function *g()* mathematically using the *max()* function over the set of 0.0 and the input *z*; for example:

g(z) = max{0, z}

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.

We compile the model using a standard ‘adam’ optimiser performing stochastic gradient descent and testing the loss by ‘binary\_crossentropy’. This is also called the log loss in classification as described in the equation below.

A close up of a logo

Description automatically generated

Binary Cross-Entropy / Log Loss

We split the dataset of 128069 records into training and test set where the training data contains 100000 records and the remaining is considered as the test data. We train the model using the train data predictors and the response variables repeating this for 1000 epochs and setting 20 % as the validation set.

Given the stochastic nature of the training algorithm, specific results may vary. In this case, we can see that the model learned the problem exceptionally well for the train data, achieving well over 98% accuracy on the training dataset and but the validation accuracy results in as low as 54 %. On further testing, we find that the model only predicted false for the validation set, confirming that the model overfit the data.

By design, the output from ReLU is unbounded in the positive domain. This means that in some cases, the output can continue to grow in size. As such, it may be a good idea to use a form of weight regularization, such as an L2 vector norm to prevent overfitting.

A close up of a logo

Description automatically generated

Due to the addition of this regularization term with lamba as the regularising parameter, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it reduces overfitting to quite an extent. We select a lambda value of 0.00001 as the kernel\_regulariser in the compile model stage and re-train the model to get to accuracy of close to 90.79 % on the test dataset.

The loss function for both validation and the train data is as shown in the graph below. There is a steep decline in loss initially and remains more or less constant as the model progresses to train for 1000 epochs.