SENG 474 Final Report

NBA Shot Expected Point Value Prediction

Michael Reiter
Faculty of Engineering
University of Victoria
Victoria, Canada
mreiter@uvic.ca

Erik Reppel
Faculty of Engineering
University of Victoria
Victoria, Canada
ereppeld@uvic.ca

Spencer Vatrt-Watts
Faculty of Engineering
University of Victoria
Victoria, Canada
asvw@uvic.ca

Chinyere Ibelegbu
Faculty of Engineering
University of Victoria
Victoria, Canada
chinyere@uvic.ca

***Abstract*—This project intends to predict the expected point value (0 - 3) of a basketball shot in an NBA game given known features. Data of shots taken during the 2014-2015 NBA season is sourced from Kaggle and includes 120,000+ data points. The group scraped further data to provide context of the player taking the shot. The data was normalized and rescaled where necessary to produce better predictions. For each model tested, the training set was shuffled and models were retrained 100 times to demonstrate training features and predicted label correlation. A Multi-layer Perceptron neural network with hidden layer shape (len(features), len(features), 5) and tanh activation function was found to be the most accurate model tested.**

Keywords—software engineering; data mining; machine learning; basketball

I. Introduction

This project intends to predict the expected point value (0 - 3) of a given a basketball shot in an NBA game. This metric allows for a clear comparison between shot classes, taking into account the quality of the shot. The dataset does not include free throws because they are far more straightforward to predict and depend on fewer variables. Since shots can be worth either 2 or 3 points, the prediction will be a value in the continuous interval between 0 and 3 inclusive. We believe it will be interesting to determine which features have the greatest effect on predicting shot quality. The core motivation behind determining expected shot value was being able to predict and compare shots before they are taken. Basketball statistical analysis is a highly monetizable area of research, so there are a great number of websites and papers devoted to it. Every year, there is a conference held in Boston called Sloan dedicated to sports analytics [1]. Since there is great profit to be made for winning in professional basketball, teams are willing to spend large sums of money on analytics that could provide them with a competitive edge.

II. Related Work

A particularly interesting piece of related work is “Psychology of a Professional Athlete”, a study on Kobe Bryant conducted by Kaggle user Selfish Gene [2]. The publicly available dataset used contains the location and attributes of all field goals Kobe attempted throughout his career. Selfish Gene worked to see if Kobe was an irrational basketball player. He attempted to prove if Kobe’s feeling of “being in the zone” actually caused him to shoot better. Unsurprisingly, the answer was no.

With regards to our problem domain, predicting expected shot value, there is very little work that has been done aside from people on Kaggle experimenting with the same dataset we worked with. There is Yisong Yue's fascinating work [3] on predicting a player’s likelihood of passing or shooting based on where they are in relation to the hoop and other players on the court. However, this is a different problem. Yue’s work deals with how likely a shot is to occur, rather than the expected point value from a given shot.

III. Data Source and Content

Data of shots taken during the 2014-2015 NBA season is sourced from Kaggle [4] and includes over 120,000 data points. This dataset is composed of the following features: shot number, period, game clock, shot clock, dribbles, touch time, shot distance, points type, closest defending player, game location, and shooting player id.

Shot number is the nth shot a player has taken in a game. Some players improve over the course of a game, while others perform worse as the game progresses. Period is the quarter of the game (1, 2, 3 or 4). This could be of interest as some players are considered “clutch”, meaning they shoot well very late in the game. Game clock indicates how much time remains in a given period. Shot clock is a 24 second countdown that resets whenever possession of the ball changes teams. Dribbles indicates the number of times the player bounced the ball before shooting. This implies movement around the court. Touch time indicates the time that has elapsed since the player took possession of the ball. Shot distance is the distance from the player shooting to the hoop. Nearer shots have a much higher probability of being made, but are limited to 2 points, while shots taken from beyond the three point line are less likely to go in, but are worth 3 points. Points type indicates whether the shot is a 2 pointer or a 3 pointer. This number sets the upper bound on expected value. Closest defending player measures the distance from the shooter to the nearest defender. Clearly a small value will decrease a shot’s expected point value since it is likely to be blocked. Game location specifies whether the game is played at home or away. Many people believe there is a home court. We do not believe this will have a significant effect on an individual shot, although it could affect shots late in the game due to crowd pressure. Shooting player id allows us to include player specific context into predictions using supplementary player data.

While the Kaggle dataset has a fair number of relevant features, it fails to take into account contextual details about the player taking the shot. Additional data was scraped from the NBA stats REST API [5]. It includes features such as player id, team id, age, player height, player weight, field goal percentage, three point percentage, field goals made, field goals attempted, three point shots made, and three point shots attempted.

We predicted that these further details about players could allow for more in depth shot analysis based on the season's trends for a given shooter. Since the player data we scraped and the original Kaggle dataset were both sourced from stats.nba.com, the player id attributes match without any manipulation.

It is important to notice that neither dataset takes free throws into consideration. Instead, the shots taken are broken down as follows. 73.5% are two pointers, while the remaining 26.5% are three pointers. Of the two pointers, 48.8% are made. Of the three pointers, 35.2% are made.

One concern the group quickly realized and addressed was greatly varying data points. For example, field goal percentage is a value constrained in the range between 0 and 1, but for a feature like number of dribbles, the player may have dribbled around the court for some time, leading to a significantly higher number. Data normalization was employed to combat this large variance between feature values. A simple feature rescaling formula (1) was applied to certain attributes in order to constrain the data range between 0 and 1.

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

(1)

The group found that normalization through feature rescaling improved prediction accuracy by between 5 to 10% on average when compared to non-normalized data on the same model.

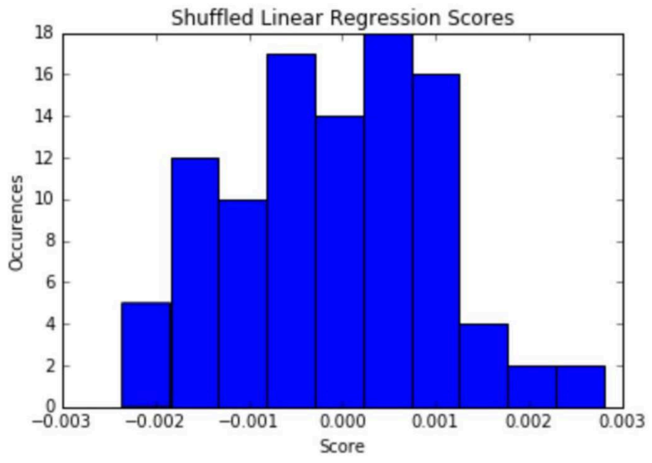
IV. Methods Used

The group trained four different models and evaluated their performance. Each model was fit and scored and this performance was recorded. The training labels were then randomly shuffled, and the models were refit and scored 100 times. The shuffled scores are plotted as histograms to illustrate the correlation between the training data and the expected point value. One would expect the unshuffled score to be clearly higher, but this was not always the case.

A. Linear Regression

Michael first trained a linear regression model [6]. Even after normalizing the dataset, the model scored only a paltry 0.03 to 0.04.

Fig. 2 below is a histogram plotting 100 scores for a Linear Regression model trained on shuffled data. As expected, the unshuffled score was markedly higher than shuffled scores at 0.035. The shuffled scores were close to zero, sometimes even negative.



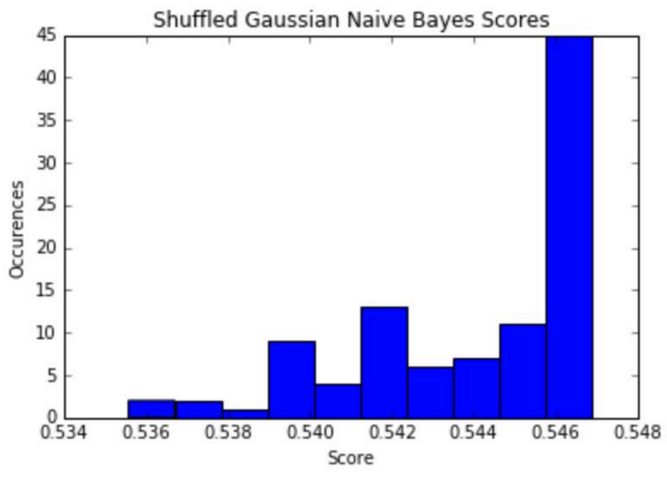
2. Shuffled Linear Regression Scores

B. Naive Bayes

After it was determined that regression models would be difficult to compare, the group decided to pivot toward classification models. Provided a model could produce the probability of choosing each label, classification could be used to compute expected value by taking the sum of each label multiplied by the probability of that label's selection.

Since the basis of Naive Bayes models is determining the labels' probability of selection, Michael elected to train and compare Gaussian and Bernoulli Naive Bayes classifiers [7]. After tuning some parameters, the Gaussian model scored a modest 0.46. Meanwhile the Bernoulli model scored around 0.53. This result was unexpected since Bernoulli assumes all features to be binary, which is not an accurate reflection of the dataset. Upon further inspection of the sklearn documentation, it appears BernoulliNB binarizes the input data for compatibility and this lead to improved results. Neither model was exceptionally accurate.

Fig. 3 below is a histogram plotting 100 scores for a Gaussian Naive Bayes classifier trained on shuffled data. Unsurprisingly, the shuffled data all performed worse than the unshuffled data.

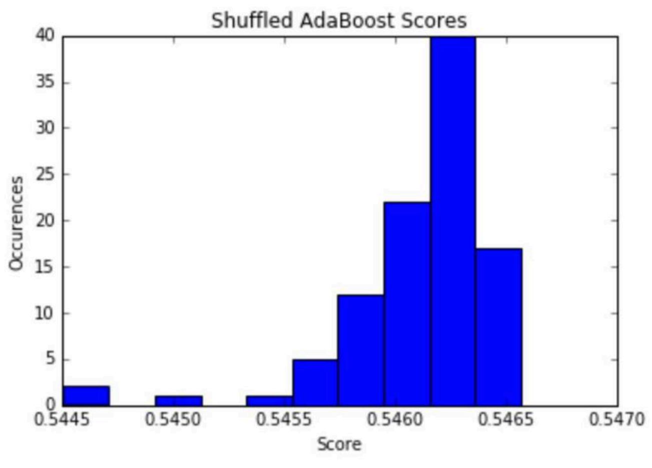


3. Shuffled Naive Bayes Scores

C. *AdaBoost*

Spencer trained a model using the AdaBoost classifier. He expected to receive strong results since AdaBoost combines many weak learning algorithms to create a final boosted classifier [8]. Because AdaBoost is sensitive to outliers in the dataset, the data was normalized before training. The best score he was able to achieve was 0.5463.

Fig. 4 below is a histogram plotting 100 scores for an AdaBoost classifier trained on shuffled data. Unshuffled AdaBoost also scored better when compared to random shuffling.

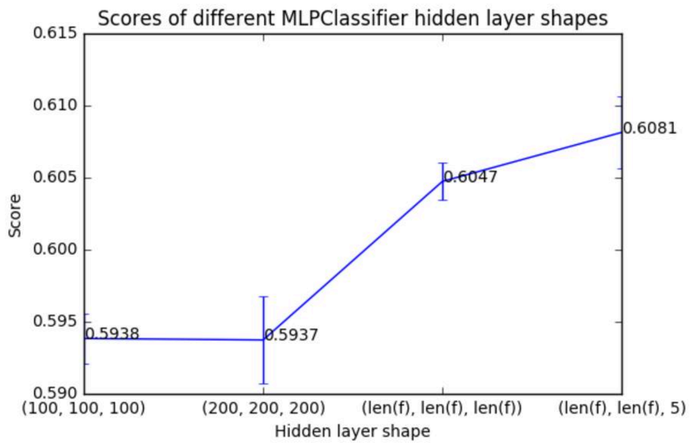


4. Shuffled AdaBoost Scores

D. *Multi-layer Perceptron*

Erik explored neural networks in an effort to improve accuracy. Using a multi-layer perceptron [9], he scored the highest of any model the group tested, 0.6179. We hypothesize that the hidden layers allow more room for error to account for edge cases. In other words, more parameters to tune means outliers in the dataset have a smaller negative effect on accuracy.

Fig. 5 below plots scores of sklearn's Multi-layer Perceptron Classifier using logistic sigmoid as an activation function and tuning the shape of the hidden layers. The x-axis shows a tuple where the ith item represents the number of neurons for the ith layer.



5. Multi-layer Perceptron layer shapes vs. scores

In the process of optimizing the multi-layer perceptron classifier, the group tried a wide variety of techniques. These can be generalized into trying to optimize the following categories: network shape, activation function, optimization function, and feature set.

Sklearn defaults the shape of its MLP classifier to (100, 100), meaning two hidden layers, each consisting of 100 neurons. Initially, we experimented with increasing the number of layers, but saw a convergence in benefit after 3 hidden layers at a score of 0.5938 +/- 0.0035. Then we attempted to increase the size of our hidden layers, using (200, 200, 200) as the hidden layer shape. This resulted in a score of 0.5937, which is well within our variance and thus cannot be considered an improvement. We decided next to move in the opposite direction, decreasing the size of the hidden layers. Using hidden layer shaped as (len(features), len(features), len(features)) produced a score of 0.6047 +/- 0.0026. This caused us to consider smaller hidden layer sizes, and eventually decided on (len(features), len(features), 5) as the most optimal hidden layers sizing with a score of 0.6081 +/- 0.0050. These sizes may seem arbitrary, but they follow the opinion of Jeff Heaton (author of Introduction to Neural Networks in Java [10]) on the size of hidden layers, who posits that the optimal hidden layer size is typically less than the size of the input layer, and greater than the size of the output layer. Given our input layer of 14 features, and our output layer of 3 classes, this shape agrees with Mr. Heaton.

The next parameter of the MLP we tried to optimize was the activation function. This is the function run on the resultant of multiplying and summing the weights and inputs for each layer and maps the value between 0 and 1, or -1 and 1. The functions supported by sklearn are “relu”, “sigmoid”, “identity” and “tanh”, with relu being the default. By testing the various functions with the hidden layer sizes we already established, we concluded that the tanh function performed the best producing a value of 0.6179 +/- 0.0042. The sigmoid function performed second best with a value of 0.6097 +/- 0.0015. We hypothesize that tanh performs best because it maps values between -1 and 1, rather than 0 and 1. Our dataset has features that have some features which intuitively have a negative impact on expected shot value, such as nearest defender distance. There are also features that are intuitively additive, such as distance to the hoop (since beyond the three point line, the shots are worth more). We feel that allowing for both negative and positive values better captures the difference in these types of features.

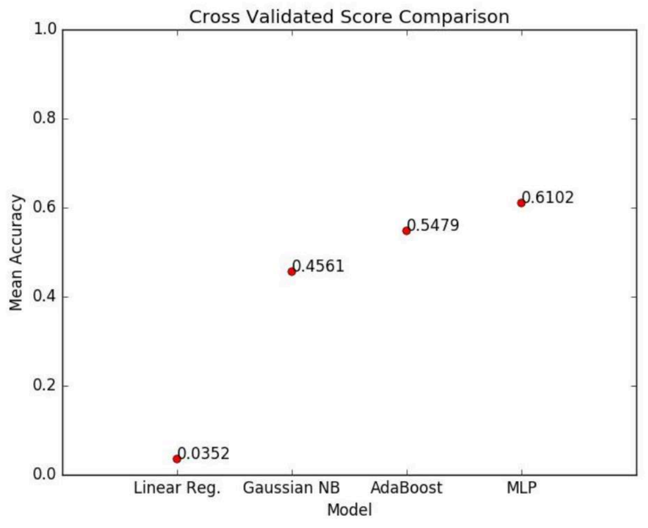
V. Results

Spencer experimented with training various models using every possible subset of the available features in an effort to determine which combination yielded the greatest accuracy. Unfortunately, this proved to be a mostly futile effort as the group’s chosen model, sklearn’s Multi-layer Perceptron, performed best when trained using the full feature set. It is clear, however, that some features appear to have a stronger correlation with shot quality than others, based on the feature subset testing that was undertaken. Notably, when the shot distance and points type features are included, the model performed approximately 5% better, while other features has a less significant impact. Training on those two features alone achieved an accuracy only about 2% lower than the entire feature set.

Shots with similar features could vary from player to player. For example, some players rarely attempt three pointers, so even if they had a high distance to the nearest defender, the expected point value would still be low. Initially, the group hypothesized that including advanced features for player context could lead to a significantly more verbose model that produces a better prediction. Surprisingly, the inclusion of these features proved to be insignificant, adding less than 1% accuracy.

The group wanted to bring more context to each shot by including which player took the shot. To achieve this, we used the one-hot method, where n columns are added to the data matrix, each column representing one of the n players. If a shot was taken by a player the feature representing that player has the value 1, otherwise it has the value 0. This means that each shot should only have 1 of the n columns containing a 1, with the rest all being 0. When we re-trained are model including the one-hot player names, shockingly the model performed significantly worse, scoring 0.5329 +/- 0.0150. This is nearly 8% worse than without the one-hot columns.

To perform cross validation, the group randomly shuffled and then split the testing dataset. Afterwards, the models were trained and tested on different parts of the random split. After repeating this process five times, the mean accuracy of each model was calculated. These results are visualized in the accompanying graph, with each model plotted against its mean accuracy.



6. Comparing accuracy of tested models

VI. Discussion of Results

In the original project proposal, the group postulated that the most accurate scores would result from a linear regression model trained using gradient descent. Conversely, it is now clear that linear regression scored significantly less than other models tested.

The very low accuracy is due to the scoring algorithm used (7). LinearRegression’s score function returns the coefficient of determination R^2 of the prediction.

$$R^2 = 1 - \frac{sum\left(\left(y_{true} - y_{\text{pred}}\right)^2\right)}{sum\left(\left(y_{true} - mean\left(y_{true}\right)\right)^2\right)} \tag{7}$$

Since linear regression produces a continuous expected point value, it often does not align well with the target labels being 0, 2 or 3. The regression did actually produce some effective predictions, but due to the method of scoring, the results appear weak. For example, if a three point shot were to have a 50% chance of being made, clearly a good expected value prediction would be about 1.5. However, if the shot were missed, the target label would be 0. Conversely, if the shot were made, the target label would be 3. Either way, the prediction of 1.5 would be off by a significant margin leading to a low perceived score. It is even possible for the score to be negative if the predictions are particularly inaccurate.

All regression models are prone to the problem of artificially low scores even if they produce good predictions in reality. In fact, a high score for us would actually indicate that we are giving a poor prediction by the way that we would intuitively score shots. For a model to achieve a perfect score, our regressor would predict the exact value for every shot. For example, two point shots that were made would produce expected values of 2.0. This implies a 100% probability of these shots being successful, which is realistically impossible to achieve and would indicate that our model is not performing well. Since it is impossible for us to know what score a regression model should produce to perform well given our criteria, regression models cannot be compared.

Comparing performance between classification models is as simple as evaluating how often the correct label is selected. These scores tend to be much higher, especially when the model is strong and there are few labels to choose from.

We predicted expected values for each shot in our test set using a model trained on both two point and three point shots. It is worth noting that all the feature values in the table are the scaled values for each feature. We also wish to highlight two three point shots, in Fig. 8. For the first shot, our expected value prediction (pred_expd_val) was 0.86, while the second shot had an expected value of 1.44. The second shot would be considered very good, as the prediction indicates that our model thinks there is a very high chance of it being successful. In basketball there is a metric called offensive efficiency. It measures how many points a team would score given 100 possessions of the ball (a rough approximation of the possessions a team may have in a game). The NBA's current leader for offensive rating, the Golden State Warriors, have a score of 113.8. This means that they score 1.138 points per possession on average. We can use this as a benchmark to when considering if a shot is good or not. As we can see, our expected value of 1.44 points for that possession means it is a very good shot - quite a bit higher than the benchmark provided by the Warriors. Looking at the features for this shot, the expected value prediction makes intuitive sense, as it has a lot of features that we would imagine a shot with a high expected value to have. For example, a large distance to closest defender, a short distance to the basket, and the shot is taken by a player who historically makes a lot of three point shots. The first shot on the other hand has a relatively poor expected value, but we also can see in Fig. 9 that it is still a better shot to take than many two point shots. The first shot's prediction also makes logical sense, as it is a three point shot with a large distance to the hoop, a defender nearby, and not much time on the shot clock remaining, leading us to think the shot would unlikely be successful.

Additionally, after separating the dataset into two point and three point shots, we found the following. When the models were trained on only the two point shots, we achieved a score of 0.6076 +/- 0.0055, which is slightly below our best score for the combined dataset (0.6174 +/- 0.0047). When the models were trained using only the three point shots, the best score achieved was stronger at 0.6487 +/- 0.0006.

If the group had more time to develop the project, we would try to incorporate other data sources to produce tangible interesting applications. Other data sources may have yet more features of interest which could contribute to model accuracy. Synergy sports technology [11] provides realtime streams of data during NBA games. Since evaluating the model to produce an expected shot value can be performed very quickly, as the ball moves around the court, we could produce a statistic indicating what the expected point value would be if a given player were to shoot at a given time.

Given more time, the group would also like to use data from other seasons as well, although this may or may not be effective because it would not account for shooting trends within a season. For example, if the entire league improved at shooting three pointers, the older worse performing seasons would bias the results.

VII. Group Member Contributions

Erik initially proposed the project subject, scraped the advanced features dataset, trained several models, tuned the multi-layer perceptron classifier, and contributed significantly to the final report.

Michael wrote the project proposal, midterm report, project presentation slides, and most of the final report. He also trained and tested the linear regression and naive bayes models.

Spencer trained the AdaBoost classifier, helped generate graphs and histograms to visualize data and wrote the related work section of the final report.

Chinyere contributed to the project presentation and final report introduction and conclusion.

	SHOT_NUMBER	PERIOD	SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	CLOSE_DEF_DIST	FG3A	FG3M	FG3_PCT	FGA	FGM	FG_PCT	PTS_TYPE	pred_expd_val	actual_value
17269	0.162162	0.500000	0.058333	0.0	0.872149	0.529661	0.080827	0.345679	0.277778	0.349	0.250000	0.264368	0.366234	3	0.867009	0.0
107786	0.162162	0.166667	0.791667	0.0	0.873210	0.476695	0.246241	0.728395	0.722222	0.437	0.519231	0.574713	0.394805	3	1.448873	3.0

8. Comparing three point shots

	SHOT_NUMBER	PERIOD	SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	CLOSE_DEF_DIST	FG3A	FG3M	FG3_PCT	FGA	FGM	FG_PCT	PTS_TYPE	pred_expd_val	actual_value
77944	0.054054	0.166667	0.000000	0.03125	0.875332	0.548729	0.084586	0.061728	0.027778	0.159	0.403846	0.459770	0.431169	3	0.478614	0.0
22933	0.081081	0.333333	0.729167	0.00000	0.873210	0.101695	0.020677	0.000000	0.000000	0.000	0.399038	0.540230	0.625974	2	1.000848	0.0
126766	0.027027	0.000000	0.120833	0.18750	0.908753	0.381356	0.067669	0.518519	0.416667	0.359	0.557692	0.563218	0.285714	2	0.707289	2.0
104190	0.378378	0.500000	0.000000	0.00000	0.871618	0.544492	0.082707	0.395062	0.333333	0.376	0.533654	0.586207	0.392208	3	0.859593	0.0
82348	0.243243	0.333333	0.620833	0.25000	0.902918	0.122881	0.048872	0.493827	0.416667	0.386	0.548077	0.563218	0.314286	2	0.941311	0.0
8540	0.000000	0.000000	0.883333	0.06250	0.887533	0.082627	0.097744	0.000000	0.000000	0.000	0.192308	0.275862	0.724675	2	1.449162	0.0
23573	0.000000	0.166667	0.429167	0.00000	0.872149	0.364407	0.077068	0.012346	0.000000	0.000	0.278846	0.298851	0.384416	2	0.766498	2.0
16789	0.027027	0.166667	1.000000	0.00000	0.873740	0.016949	0.016917	0.000000	0.000000	0.000	0.197115	0.264368	0.618182	2	1.170582	2.0
2683	0.162162	0.166667	0.841667	0.03125	0.874801	0.008475	0.075188	0.555556	0.388889	0.304	0.701923	0.620690	0.155844	2	1.616030	2.0
80030	0.216216	0.500000	0.454167	0.00000	0.872149	0.103814	0.043233	0.000000	0.000000	0.000	0.153846	0.172414	0.475325	2	1.026017	2.0
67386	0.108108	0.333333	0.916667	0.03125	0.874801	0.508475	0.086466	0.641975	0.527778	0.354	0.447115	0.425287	0.236364	3	1.115874	0.0
102002	0.081081	0.500000	0.366667	0.03125	0.877454	0.120763	0.013158	0.160494	0.138889	0.402	0.250000	0.275862	0.425974	2	0.855358	0.0
651	0.378378	0.500000	1.000000	0.00000	0.867905	0.055085	0.009398	0.012346	0.000000	0.400	0.687500	0.781609	0.405195	2	1.120877	2.0
62663	0.054054	0.166667	0.475000	0.09375	0.887003	0.019068	0.013158	0.333333	0.250000	0.341	0.533654	0.540230	0.288312	2	1.066637	2.0
55587	0.081081	0.500000	0.500000	0.00000	0.882228	0.296610	0.084586	0.049383	0.027778	0.281	0.341346	0.402299	0.464935	2	0.849908	2.0
317	0.054054	0.500000	0.670833	0.25000	0.907692	0.474576	0.092105	0.333333	0.250000	0.321	0.245192	0.195402	0.166234	2	0.775979	0.0
38788	0.270270	0.333333	0.550000	0.09375	0.889125	0.086864	0.031955	0.654321	0.555556	0.380	0.548077	0.586207	0.355844	2	1.022627	0.0
39033	0.162162	0.333333	0.304167	0.00000	0.872679	0.466102	0.080827	0.654321	0.555556	0.380	0.548077	0.586207	0.355844	3	1.222631	0.0
17269	0.162162	0.500000	0.058333	0.00000	0.872149	0.529661	0.080827	0.345679	0.277778	0.349	0.250000	0.264368	0.366234	3	0.867009	0.0
107786	0.162162	0.166667	0.791667	0.00000	0.873210	0.476695	0.246241	0.728395	0.722222	0.437	0.519231	0.574713	0.394805	3	1.448873	3.0

9. Comparing two point shots

References

1. "MIT Sloan Sports Analytics Conference", MIT Sloan Analytics Conference, 2016. [Online]. Available: http://www.sloansportsconference.com. [Accessed: 06- Nov- 2016].
2. "Kobe Bryant Shot Selection | Kaggle", Kaggle.com, 2016. [Online]. Available: https://www.kaggle.com/selfishgene/kobe-bryant-shot-selection/psychology-of-a-professional-athlete/comments. [Accessed: 02- Dec- 2016].
3. Y. Yue, P. Lucey, P. Carr, A. Bialkowski and I. Matthews, "Basketball Pass & Shot Prediction", Projects.yisongyue.com, 2016. [Online]. Available: http://projects.yisongyue.com/bballpredict/ [Accessed: 06- Nov- 2016].
4. "NBA shot logs | Kaggle", *Kaggle.com*, 2016. [Online]. Available: https://www.kaggle.com/dansbecker/nba-shot-logs. [Accessed: 01- Dec- 2016].
5. "NBA.com/Stats", Stats.nba.com, 2016. [Online]. Available: http://stats.nba.com. [Accessed: 03-Nov-2016].
6. "sklearn.linear_model.LinearRegression — scikit-learn 0.18 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. [Accessed: 05- Nov- 2016].
7. "sklearn.naive_bayes.GaussianNB — scikit-learn 0.18 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. [Accessed: 05- Nov-2016].
8. "sklearn.ensemble.AdaBoostClassifier — scikit-learn 0.18 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html [Accessed: 05- Nov- 2016].
9. "sklearn.neural_network.MLPClassifier — scikit-learn 0.18 documentation", Scikit-learn.org, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. [Accessed: 05- Nov- 2016].
10. J. Heaton, Introduction to Neural Networks for Java, 1st ed. St. Louis, Mo.: Heaton research, 2008.
11. Synergy Sports Technology, 2016. [Online]. Available: https://corp.synergysportstech.com. [Accessed: 02- Dec- 2016].