The National Basketball Association (NBA) Prediction

Shaneer Luchman, Daniel Mundell, and Akaylan Perumal

University of KwaZulu-Natal, South Africa
{216003502,220108104,216035695}@stu.ukzn.ac.za

Abstract
In this research paper we investigated how accurately we could detect outstanding NBA players(anomalies) and predict NBA game outcomes with a variety of machine learning techniques. We aimed to determine which performed better between supervised anomaly detection techniques and unsupervised anomaly detection techniques. Our Decision tree and Support Vector machine achieved a 100% outlier detection rate. For our game outcome prediction, we achieved a maximum accuracy of 68.43% with a simple Neural Network, 68.17% with Linear Regression, and 67.74% with Logistic Regression, while also comparing to a Support Vector Machine, k-Nearest Neighbours and Random Forests.

1. Introduction

On average, each NBA team was worth $2.12 billion in the 2018-19 season [1]. It is therefore important that the teams give high paying contracts to, and make use of, the best available players in order to increase their chances of winning games and whole seasons. Because basketball is a complex sport, and there are many ways that experts might decide who the best players are, we decided to investigate whether machine learning techniques could be used to identify those players which show exceptional performance. This could potentially help team owners find up and coming exceptional players who have not yet been identified by experts and other teams. We created three supervised learning models; a Naive Bayes Classifier, a Support Vector Machine and a Decision Tree to compare how these three model's perform for anomaly detection. We also implemented two unsupervised learning models; an Isolation Forest and a Local Outlier Factor, as our focus was how supervised anomaly detection compares to unsupervised anomaly detection. The supervised algorithms' training data will be labeled so the outliers are known. On the other hand, the sports bettings industry was valued at an extraordinary $104.31 billion in 2017, and is expected to grow upwards of $155.49 billion by 2024 [2]. Using machine learning techniques to predict game outcomes for sports betting could therefore be an extremely profitable business, if a high enough accuracy rate was achieved. Because of this, we also decided to also compare the performance of various machine learning methods when used to predict NBA game outcomes. We implemented six supervised learning models, including Linear Regression, Logistic Regression, Support Vector Machine, Neural Network, Random Forests, and k-Nearest Neighbours.
Our project code is available at: https://github.com/danielukzn/Comp721_NBA_Prediction

2. Related Work

Various research has already been conducted on the topics of outlier detection and game outcome prediction in regards to the NBA. In 2009, M. Beckler, H. Wang, and M. Papamichael applied linear regression, logistic regression, a support vector machine (SVM), and an artificial neural network (ANN) to the NBA game outcome prediction problem and were able to achieve up to 73% accuracy, while most accuracy levels were between 62-70%. In their testing, linear regression performed the best, and ANN the worst. They also created a scatter plot based on a linear combination of features to visually detect outstanding player outliers, and were able to detect the majority of NBA all-star players, but did not share any accuracy rates [3].

In 2017, J. Uudmae predicted NBA game outcomes and game scores based on previous game scores and home/away advantage over the period 2013-2016 using SVM, linear regression and neural network regression. They achieved game outcome accuracy between 62-65%, and were between an average of 10-12 points off per game. Their neural network regression provided the best accuracy [4].

In [5] the anomaly detection problem was formulated as a continuous stream of data points built from a collection of measurements governed by some probability distribution to detect anomalies in a network. Multiple techniques were used. The first of which is a One-Class Neighbour Machine (OCNM). This algorithm is used to identify points that lie within the minimum volume set with the smallest sparsity measure that is required for the algorithm. The next algorithm is Kernel-based Online Anomaly Detection (KOAD). The KOAD algorithm uses a dictionary of approximately linearly independent elements to save computational time and storage space. KOAD works with finding a projection error for a measurement vector at a particular timestep. This error value is compared with two threshold values to determine whether the measurement vector represents normal behaviour. This method can differentiate between anomalous data and changes in the normal behaviour of data. Lastly, a Monitoring architecture was explored. [5] Used a distributed approach and a centralized approach in which each node makes a local decision with respect to the presence of an anomaly. Based on the results obtained in [5] the KOAD algorithm is a better option in this case due to its recursive nature and ability to adapt to changing data streams.

In [6] anomaly detection systems were considered an alternative to signature-based intrusion detection systems. [6] goes on to provide descriptions of multiple anomaly detection techniques. These techniques can either be supervised or unsupervised. In the case of supervised techniques, we have options such as K-Nearest Neighbour, Bayesian Network, a supervised Neural Network, Decision Trees and Support Vector Machines. Supervised training requires a labelled training data set of both anomalous and regular samples. In the case of unsupervised learning, the options listed are an unsupervised Neural Network, K-Means, Fuzzy C-Means, Unsupervised Niche Clustering, Expectation-Maximization Meta and One-Class Support Vector Machine. These do not need training data and assume that large clusters of data represent normal behaviour. Pros and cons were listed for the aforementioned anomaly detection system. It was found that supervised learning methods performed better and among the supervised learning methods, non-linear methods performed the best (SVM for example).

3. Methods and Techniques

3.1 Outlier Detection

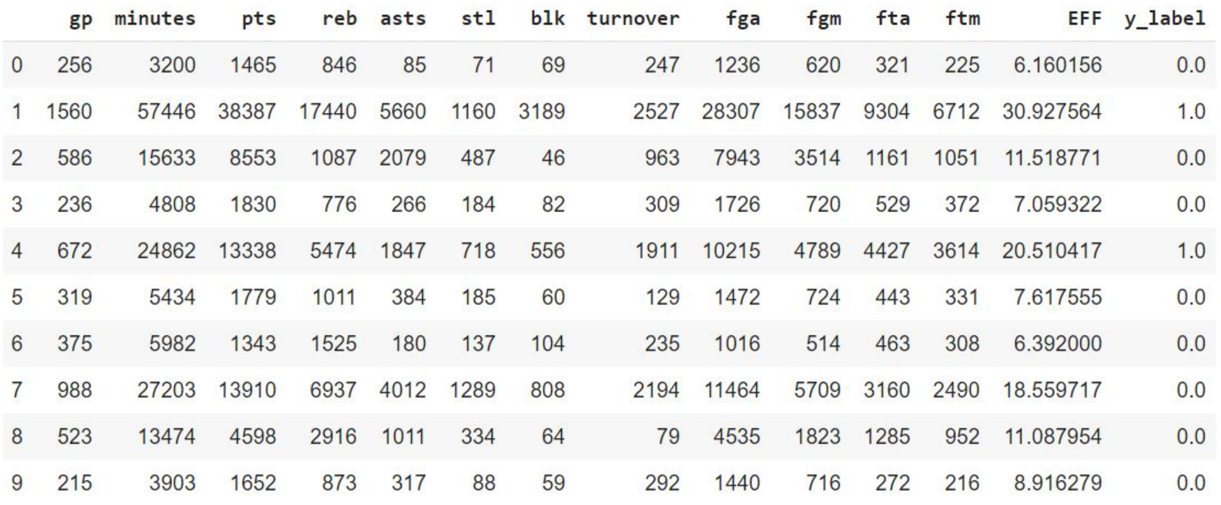
A) Dataset

We were provided a dataset of basketball statistics [7] covering a variety of player and team stats for the period 1946 - 2004. To perform our task of determining the outstanding players of the NBA/ABA history, we chose to use the "player_regular_season_career.txt" dataset. This dataset originally consisted of 21 features, recording data for each player's career stats such as their playerID, name, league played in, games played, minutes played, total points scored, offensive rebounds, defensive rebounds, total rebounds, assists, steals, blocks, turnovers, personal fouls received, field goals attempted, field goals made, free throws attempted, free throws made, three pointers attempted, and three pointers attempted made. As a pre-processing step, we removed all instances that had less than 500 minutes played as well as all instances that had less than 100 games played. This is because it would be unfair to judge if a player was outstanding or not at only the start of their careers. How do we judge if a player is outstanding or not? Martin Manley, a sports reporter and statistician developed a statistical benchmark(EFF) for comparing the overall efficiency of the players [8]. The formula is derived as follows :

$$\mathbf{(PTS + REB + AST + STL + BLK) - (TO + MissedFG + MissedFT)/GP}$$

where MissedFG = FGA - FGM and MissedFT = FTA - FTM.

We removed all columns that were not part of the formula to improve model efficiency and accuracy. We then calculated the efficiency rating for each instance using the EFF formula and added that column to the dataset. Using these values we were able to determine the ground truth labels for each instance. We classified the top 100 players, meaning values greater than 20.18 as outstanding players. Each instance was then allocated a y_label such that 0 represented a standard player and 1 represented an outstanding player. The final dataset consisted of 1902 instances and 13 features which was used for training as shown in figure 1.



	gp	minutes	pts	reb	asts	stl	blk	turnover	fga	fgm	fta	ftm	EFF	y_label
0	256	3200	1465	846	85	71	69	247	1236	620	321	225	6.160156	0.0
1	1560	57446	38387	17440	5660	1160	3189	2527	28307	15837	9304	6712	30.927564	1.0
2	586	15633	8553	1087	2079	487	46	963	7943	3514	1161	1051	11.518771	0.0
3	236	4808	1830	776	266	184	82	309	1726	720	529	372	7.059322	0.0
4	672	24862	13338	5474	1847	718	556	1911	10215	4789	4427	3614	20.510417	1.0
5	319	5434	1779	1011	384	185	60	129	1472	724	443	331	7.617555	0.0
6	375	5982	1343	1525	180	137	104	235	1016	514	463	308	6.392000	0.0
7	988	27203	13910	6937	4012	1289	808	2194	11464	5709	3160	2490	18.559717	0.0
8	523	13474	4598	2916	1011	334	64	79	4535	1823	1285	952	11.087954	0.0
9	215	3903	1652	873	317	88	59	292	1440	716	272	216	8.916279	0.0

Figure 1: Training data used on all classifiers

B) Techniques Used

We used the test_train_split function from sklearn to split the training and testing data. We chose to allocate 30% of the data for testing. For the unsupervised models we trained the models on the training data without labels, used the test data to make predictions, and then compared the predictions to the ground truth labels to assess the accuracy of the model.

Decision Tree

We implemented a Decision Tree classifier as part of the supervised training models. A decision tree is a tree that is made of nodes, leaves and arcs. A node represents the most informative feature attribute that is yet to be explored. Each leaf represents a class, and each arc represents a feature value for the node it leaves. Classification occurs by starting at the root node of the tree and traversing through until a leaf node is reached. The class that the leaf node represents is the classification decision. We conducted experiments on both criteria; Gini Index and Entropy. We set the max_depth=3 and the random_state to 100. The model was trained using the *model.fit(X_train, y_train)* and predictions were made using *model.predict(X_test*).

Support Vector Machine

We implemented a support vector machine(explained in detail in 3.2) for the supervised outlier detection section as well. SVM's map input vectors that consist of n number of features into an n-dimensional feature space. A separating hyper-plane is used as a decision boundary. This classification method is robust to outliers since the decision boundary is dependent on the support vectors instead of the entirety of the training data. We created several models by tuning the hyperparameters of the SVM to examine the effect it would have for detecting outliers. We used the values [0.01, 0.02, 0.03, 0.04] as combinations for the C and gamma parameters. The same training and predicting methods were used as the Decision Tree classifier.

Naive Bayes

The last supervised learning model we implemented was the Naive Bayes classifier. This classification method is probabilistic in nature. It utilizes assumptions of independence between features. We modified the *var_smoothing* hyperparameter in sklearn to test and compare results.

Isolation Forest

The 1st unsupervised model we trained was the Isolation Forest. This unsupervised learning algorithm is based on isolating outliers. Since the focus is placed on anomalous data points, the algorithm is very fast and does not require as much memory since sampling techniques can be used to reduce sample size. A training dataset is used to build trees that will assign a value, if this value is higher than a defined threshold then that point is considered an anomaly or outlier.
We experimented with different contamination values to determine how it affects the models ability to detect outliers. We used each of the values [0.1, 0.2, 0.04, 0.05] to train and test the model. Model testing for our unsupervised training models required an extra step where we had to convert our binary *y_labels* to either 1(inliers) or -1(outliers).

Local Outlier Factor

Another unsupervised outlier detection method we implemented was the Local Outlier Factor, which uses a density-based approach. This unsupervised learning algorithm deals with calculating the density of a particular data point and comparing that density value to its neighbour's values. If a points density value is much lower than that of its neighbours, then that point is considered an outlier.

3.2 Game Outcome Prediction

A) Dataset

We supplemented the provided dataset with schedule and results statistics for the period 1999 - 2004, also obtained from basketball-reference.com [7]. We made use of the team season and results subsets, for only NBA teams playing during the 1999 - 2004 seasons. We removed all games involving 4 teams (Charlotte Hornets, Memphis Grizzlies, New Orleans Hornets, and Vancouver Grizzlies), as they did not participate in all 5 chosen seasons. We processed each season separately, to allow us to predict the next season's game outcomes based on the current season's data. We scanned through the results dataset, and created a feature vector by concatenating the team statistics for the visitor and home teams, based on the two teams in each game and the seasonal statistic available in the team season dataset. This gave us a feature vector containing 66 unique features (33 from each team). The results dataset was also used to create the corresponding labels for each game. For the neural network, the labels were an array of the two teams' points for the game. The linear regression and k-NN algorithms used the absolute difference between the points as labels, and the logistic regression, support vector machine, and random forests used binary labels, where 1 and 0 are a win and loss for the visitor, respectively.

Normalization was applied to the 1x66 feature vector using min-max feature scaling shown in equation 1. This lowers the standard deviation of each feature, and prevents large feature values from being considered more important than others by default. The algorithm weights can then properly perform their job of determining the true importance of each feature. While we manually created the outlier prediction "score" formula, for the game outcome prediction section we let the machine learning algorithms find appropriate formulas. Through iterative training, the weights of our models became the formulas which calculated and compared hidden "scores" for the two teams.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

B) Techniques Used

Multiple Linear Regression

Linear regression looks to model the linear relationship between two features. Since we had multiple features, we made use of multiple linear regression which analyses multiple independent features in order to find a linear relationship. The Y labels for our linear regression model were the difference in points between the two teams per game, as shown in equation 2 where v and h are the points for the visitor and home teams, respectively. This allowed the model to take full advantage of the continuous nature of the algorithm, and all the information available to it. If the model was instead looking to predict only a binary win (1) or loss (0), it may struggle to fit the data accurately. Equation 3 shows how the weights β_i, $i = 0, .., n$ are used to train the model to predict the absolute difference, based on the sum of each feature x_i multiplied by its corresponding weight β_i.

$$Y = v - h \tag{2}$$

$$\widehat{y} = \beta_0 + \sum_{i=1}^{n} \beta_i * x_i \tag{3}$$

Logistic Regression

Logistic regression is a supervised machine learning method which uses a logistic function to fit the training data. Initially, we tried to implement this method using the difference between points as the *Y* labels, as we did for linear regression, but found much better results when predicting a binary win for loss for the visitor team, as shown in equation 5. This is because, while linear regression works well with continuous outputs, logistic regression is mainly used for classification problems. Equation 5 shows the logistic regression's prediction function, where *e* is the base of the natural logarithm and *a* and *b* are parameters similar to the weights in linear regression. It is possible to cause overfitting with logistic regression if too many features are used with this method, but only with a much higher ratio of features to training examples.

$$Y = \begin{cases} 1 & if\ v > h \\ 0 & otherwise \end{cases} \tag{4}$$

$$\widehat{y} = \frac{e^{a+bX}}{1+e^{a+bX}} \tag{5}$$

Support Vector Machine

Support vector machines (SVM) are used to classify feature vectors into one of two categories, based on where the feature vector lies in comparison to the maximum-margin hyperplane. To fit our data to this binary output model, we placed all 66 features (33 per team) in the feature vector, and had the SVM predict the binary win or loss of the visitor team, shown previously in equation 4. Through manual testing, we found that a regularization parameter of C=2 and default radial basis function kernel provided the most consistent results.

Random Forests

Random forests is a decision tree based ensemble learning algorithm. It combines many uncorrelated decision trees to outperform any of the individual decision trees while avoiding overfitting. We made use of the scikit-learn python library's random forests classifier with 100 trees, a maximum tree depth of 5, and using the Gini impurity function to measure the quality of splits, as this seemed to give good results without extensive time or memory use. Since we used the classifier version of random forests (and not the regression version), we trained with and predicted a binary *Y* label, shown in equation 4.

Neural Network

We created a simple neural network (NN) using Tensorflow and Keras in Python. Our NN had an input layer accepting a feature vector containing all 66 features (33 per team), just one hidden layer with 10 nodes, and an output layer providing the two output values which are estimates of the points obtained by the two teams. We started with a more complex NN consisting of three hidden layers with 30 nodes each, and through iterative testing kept decreasing these numbers as we saw better results and less overfitting. We used the Rectified Linear Unit (ReLU) activation function, as this helps prevent vanishing gradients and ensures that our outputs are always positive (since basketball game points are always positive). The ReLU function also provides time-efficient performance due to its simple description, shown in equation 6. We used the Root Mean Squares Propagation (RMSProp) optimizer to help avoid exploding of large gradients and vanishing of small gradients, as well as the Mean Squares Error (MSE) loss function. We used 15% of the training data for a validation set when training over the 500 epochs, to allow for more in-depth performance analytics. Our NN's performance can be evaluated based on the difference between the predicted and true points per game, as well as whether the correct winner was predicted. We could deduce which team won based on which team had more points.

$$\widehat{y} = max(0, z) \tag{6}$$

K-Nearest Neighbours Regression

K-NN regression can be used to estimate continuous variables by calculating the weighted average of the inverse distance of the k-NN. We used the scikit-learn python library's k-NN regression implementation, using the Euclidean distance, between the 5 nearest neighbours (k=5). The *Y* labels used in this model were once again the difference in points between the two teams per game, as shown previously in equation 2. This gives the algorithm additional information for a more meaningful loss function calculation and therefore allows the algorithm to fit the data better than if predicting binary *Y* labels, as shown in equation 4.

4. Results and Discussion

Outlier Detection

All learning models were trained and tested on the same data, with the test data consisting of 543 inliers and 28 outliers. We implemented 4 variations of the Support Vector Machine, tuning the hyperparameters in each to improve results. We also changed the kernel type of the SVM. The "linear" and "polynomial" kernels produced similar results whilst the "RBF" and "sigmoid" kernels gave 0% for outlier accuracy, supporting the theory that our data is linearly separable. Table 1 shows the results for the different SVM's we created. We compared the confusion matrices, the model accuracy and the model outlier accuracy.

We noticed that regularization parameter(C) and gamma did not improve results for C > 0.002 and gamma > 0.002. We were satisfied with this as the model was able to predict all outliers correctly. Table 2 shows the results for the remaining 2 supervised learning models.

From Table 1 and 2, we note that the Decision Tree performed the best by correctly classifying every instance. The Naive Bayes classifier only predicted 68% of the outliers correctly which was the lowest value from all the supervised learning models. We plotted the predicted data for all models using the python library *matplotlib*. Figure 2 shows the results for each supervised learning model with the inliers being represented by blue circles and outliers represented by red crosses.

Table 1: Supervised Support Vector Machine results

	Confusion Matrix	Accuracy	Outlier Accuracy

Kernel = Linear
C = 0,001
Gamma = 0.001
	0 1
	[0[541 2]]
	[1[2 26]]
		98%	93%

Kernel = Linear
C = 0,003
Gamma = 0.001
	0 1
	[0[541 2]]
	[1[1 27]]
		99%	97%

Kernel = Linear
C = 0,002
Gamma = 0.002
	0 1
	[0[541 2]]
	[1[0 28]]
		99%	100%

Kernel = Linear
C = 0,002
Gamma = 0.004
	0 1
	[0[541 2]]
	[1[0 28]]
		99%	100%

Table 2 : Results for the Naive Bayes and Decision Tree

	Confusion Matrix	Accuracy	Outlier Accuracy
Naive Bayes Classifier	0 1 [0[467 47]] [1[9 19]]	90%	68%
Decision Tree(max_depth=1)	0 1 [0[543 0]] [1[0 28]]	100%	100%

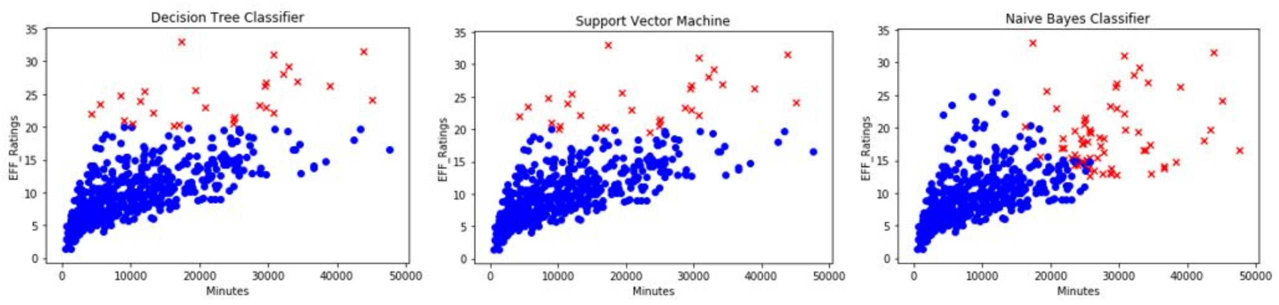


Figure 2 : Showing the outliers detected for each supervised learning model

For our 2 unsupervised learning models, we modified the contamination parameter to determine how it would affect the outlier detection. Table 3 shows the results for the Local Outlier Factor. We compared the predicted outliers with the converted ground truth labels and used that to calculate the confusion matrices. Table 4 shows the results for the Isolation Forest.

For our 2 unsupervised learning models, we noted that low contamination values gave a higher accuracy for detecting inliers, and higher contamination values gave a higher accuracy for detecting outliers, however this also became a negative as the model started predicting outliers that were not. For the Isolation Forest, a contamination value of 0.01 means we do not incorrectly predict inliers to be outliers, but in the same instance our outlier accuracy is low. Using a contamination value of 0.13 gave a higher accuracy for the outliers, but started to classify more average players as outstanding.

Table 3: Showing the results for the Unsupervised Local Outlier Factor

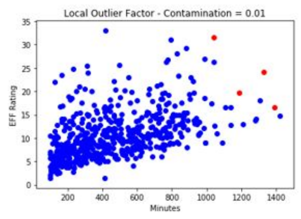
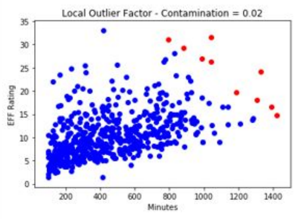
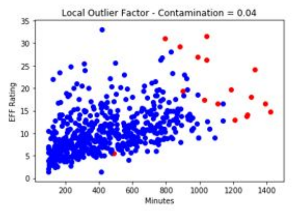
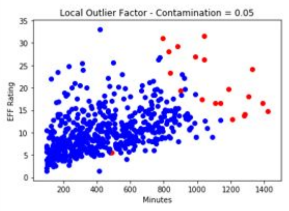
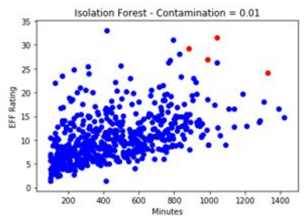
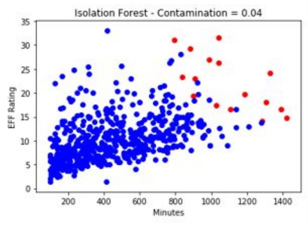
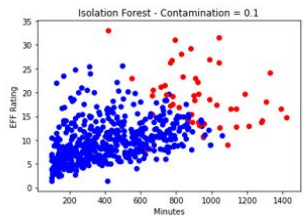
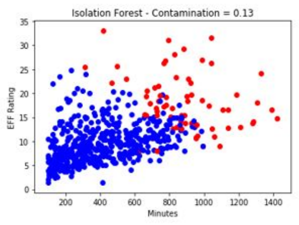
	Contamination = 0.01	Contamination = 0.02	Contamination = 0.04	Contamination = 0.05
Confusion Matrix	-1 1 [-1[2 26]] [1 [2 541]]	-1 1 [-1[6 22]] [1 [4 539]]	-1 1 [-1[6 22]] [1 [11 532]]	-1 1 [-1[8 20]] [1 [12 531]]
Plot : EFF_rating vs Minutes_played				

Table 4: Showing the results for the Unsupervised Isolation Forest Classifier

	Contamination = 0.01	Contamination = 0.04	Contamination = 0.1	Contamination = 0.13
Confusion Matrix	-1 1 [-1[4 24]] [1 [0 543]]	-1 1 [-1[8 20]] [1 [8 535]]	-1 1 [-1[17 11]] [1 [32 511]]	-1 1 [-1[20 8]] [1 [12 531]]
Plot : EFF_rating vs Minutes_played				

Game Outcome Prediction

The game outcome prediction accuracy rate for the six implemented machine learning methods has been summarized in table 5 and displayed graphically in figure 3. We limited the vertical axis of figure 3 to 60-70% accuracy to make it easier to view, although it must be noted that this pushed the last two periods for the linear regression model out the bottom of the graph. The neural network produced the best overall accuracy (up to 68.43%), although it had a much higher computational time than the other methods, shown in table 6 and figures 4 and 5, and it struggled to provide consistent results due to its stochastic nature and variation in the split between the training and validation set. While all the other methods provide the same results every time they are run on the same dataset, we had to run the neural network multiple times while assessing the validation accuracy before providing the testing dataset. Linear regression had the second best accuracy for the larger datasets (up to 68.17%), but performed worse than random guessing when trained with smaller datasets (41.91% and 42.87%). This leads us to believe that the more recent data (from 2002 and 2003) may not be as linearly related to the game outcomes, which could be caused by a number of factors, such as players or coaches changing teams during these seasons. However, due to this method's good performance for larger datasets and very impressive training and testing times, it would definitely be worth further investigation. Logistic regression had the second best overall accuracy (up to 67.74%), narrowly underperforming linear regression for the large datasets, but remaining much more consistent with the smaller datasets. This method maintained extremely low training and testing times, and would therefore be our overall best method if we needed to choose based on accuracy and computational times. Random forests was the next best method (up to 67.48%), seeing a drop in accuracy for the smaller datasets, and maintaining very low training times albeit with moderately high testing times. Its training and testing times did not increase much when the size of the dataset increased, and therefore this might still be a good method to use with very large training datasets. The SVM had very consistent accuracy rates regardless of the training dataset size (up to 66.35%). While its training times were the second worst to only the neural network, they were still acceptably low. The testing time for the SVM was impressively low, and did not depend on the dataset size, as each testing feature vector only needs to be compared to the single hyperplane in order to determine the outcome, regardless of the number of training examples. The k-NN requires very little training and testing time, but struggles to achieve near the same accuracy as the other methods (maximum of 64.35%). We tried various changes to the method parameters but could not improve its accuracy.

We could potentially improve the accuracy of all these methods by also considering the current season's past games, as well as each players' statistics for both teams. This could benefit the methods' accuracy by having access to the most up-to-date information, as it is possible that there are player or coach changes from game to game or season to season. Training the methods with a much larger dataset (such as all 1970-2003 seasons) could also help to prevent overfitting to a single season's data, and rather find a more accurate generalization to translate team statistics into game outcomes.

Table 5: 2004 Game prediction accuracy over different training periods

	1999-2003	2000-2003	2001-2003	2002-2003	Only 2003
Neural network	68.09	67.89	68.43	68.26	66.09
Linear regression	67.57	68.17	66.43	41.91	42.87
Logistic regression	67.30	67.74	67.13	66.26	66.26
k-NN	64.35	62.70	63.74	62.35	62.17
SVM	65.91	65.57	65.30	66.35	65.83
Random forests	67.30	67.13	67.48	65.74	64.61

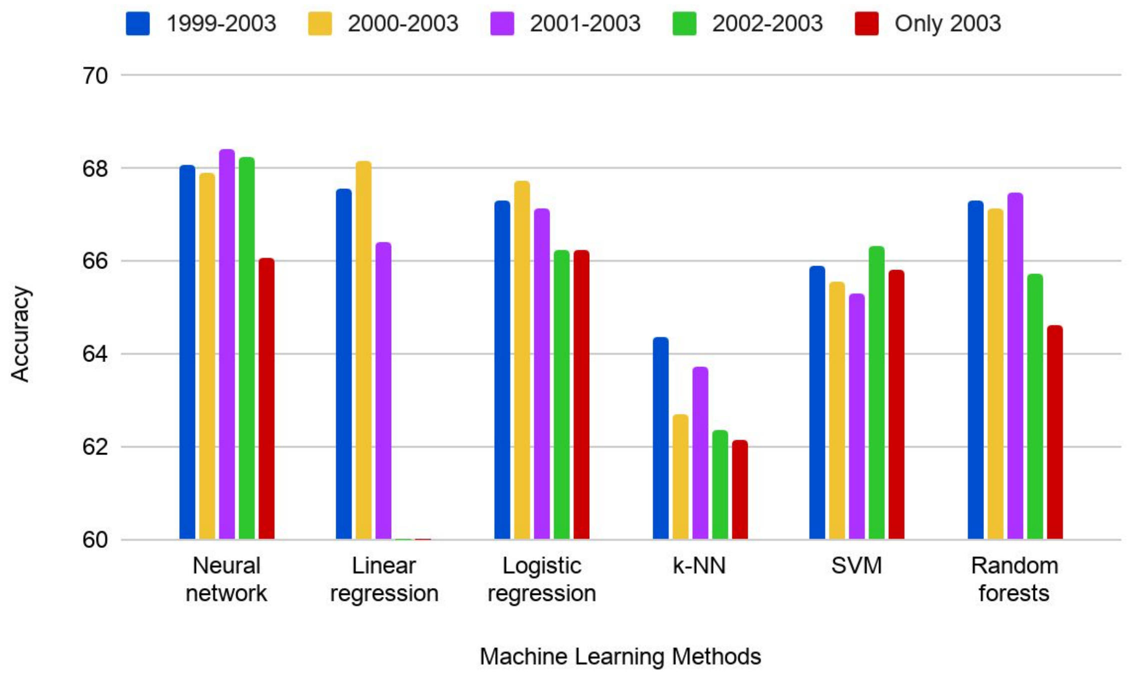


Figure 3: 2004 Game prediction accuracy for difference training periods

Table 6: Training and Testing time (ms) for different training periods

	1999-2003		2000-2003		2001-2003		2002-2003		Only 2003	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Neural network	29172	30649	25365	33170	19647	30876	14864	29593	10107	29388
Linear regression	12	60	12	65	9	43	8	46	6	44
Logistic regression	89	57	56	65	64	66	45	70	43	64
k-NN	58	929	46	868	28	708	19	580	8	455
SVM	1682	329	1066	278	609	236	261	172	63	122
Random forests	433	5997	355	6108	288	6227	201	6146	135	6109

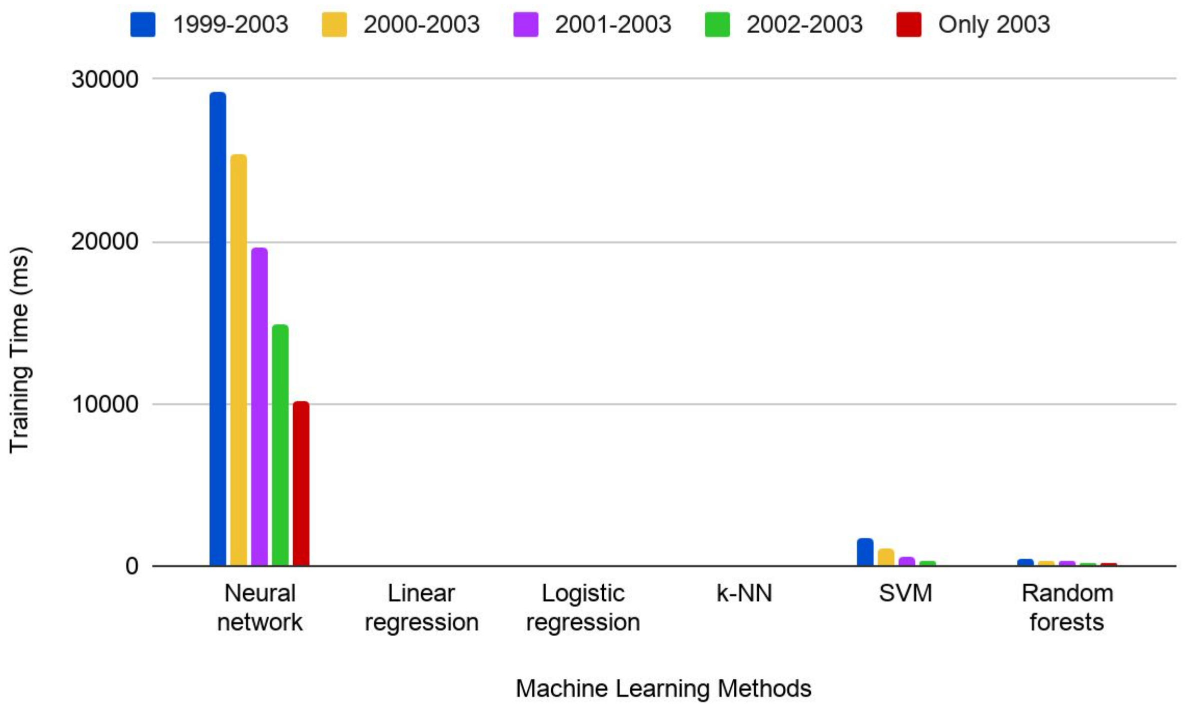


Figure 4: Training time (ms) for different training periods

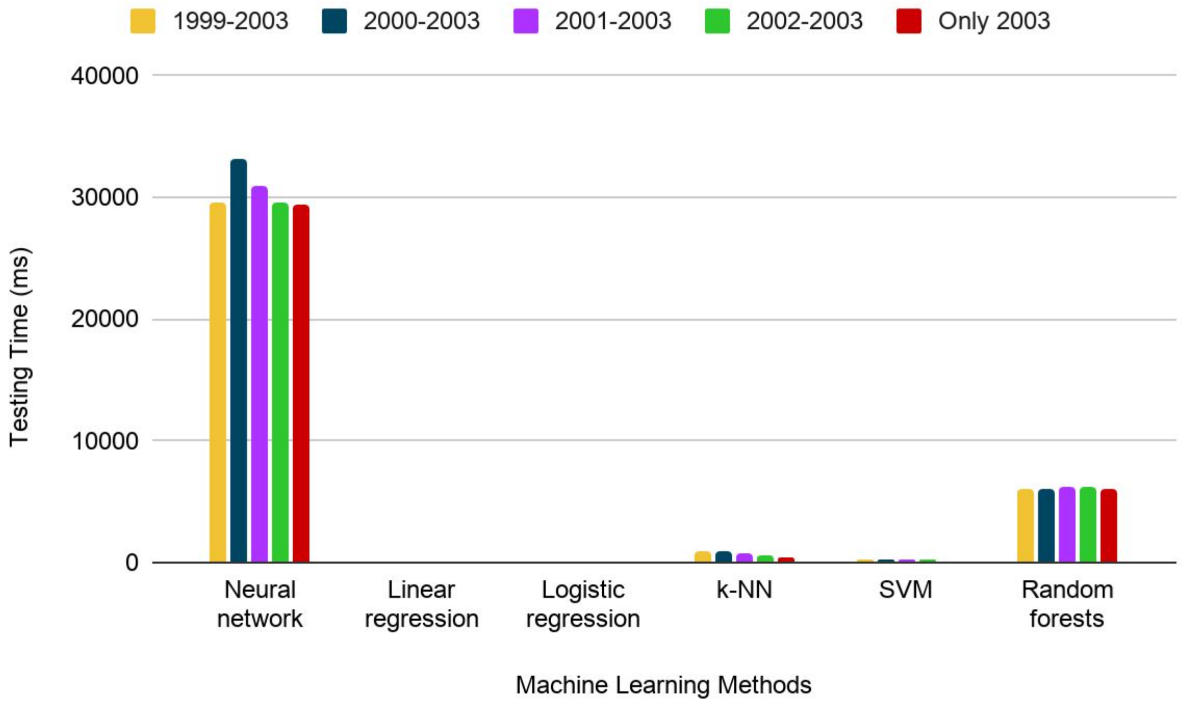


Figure 5: Testing time (ms) for different training periods

5. Conclusion

Outlier Detection

In conclusion, we answered our main question of whether supervised or unsupervised anomaly detection techniques work best. The supervised techniques outperformed the unsupervised techniques. The decision tree performed the best with a 100% accuracy for classifying both inliers and outliers. We also changed the decision tree criterion (Gini Index vs Entropy), but this had no bearing on our results. The decision tree also has the lowest computational time. The supervised Support Vector Machine performed the best when C = 0.02 and gamma = 0.02 with a model accuracy of 99% but an outlier accuracy of 1. This was deemed to be satisfactory as the main aim was to correctly predict all outstanding players. The Gaussian Naive Bayes produced the least valuable results from the supervised models, but this could be because the features on the dataset are not independent, and this particular classifier assumes that they are. The unsupervised anomaly detection models were implemented just as a comparison with the supervised methods, but we do note that the contamination parameter highly affects the way the algorithm identifies outliers. We also learned that the data we used affected the performance the most. We saw how important it was to pre-process our data and clean it, so the models can make the most out of the data. Future work could include implementing ensemble methods for the outlier prediction. We can also use different formulas to rank the players. For this paper we focused on the efficiency rating(EFF), but there are other benchmarks, such as the Defensive Player Rating(DPR) and Player Efficiency Rating(PER)[8], which could yield better results.

Game Outcome Prediction

While the neural network achieved the best accuracy of 68.43%, it was inconsistent in training and had extremely high computational times despite being a very simple network. The logistic regression method is therefore the overall best performer, for providing consistently high accuracy (66.26% - 67.74%) and extremely low computational times. There were somewhat disappointing results from the linear regression, as this method was praised by other researchers [3], but this could well be due to changes in the basketball teams across the low accuracy seasons which were not captured in the features we included. The k-NN is notably the worst performer in terms of accuracy, and is therefore probably not suited to this type of problem. In the future, we would like to consider additional features, such as player and coach statistics, as well as the game outcomes from the current season's completed games. If we decided to also greatly expand the amount of historical seasons' training data to use, the low computational times of logistic regression would make it much more suitable than the neural network, despite the slight loss in accuracy.

References

1. N. Reiff. (2020). How The NBA Makes Money. Investopedia. [online] Available at: https://www.investopedia.com/articles/personal-finance/071415/how-nba-makes-money.asp. Accessed: 9 December 2020.
2. Zion Market Research. (2019). US $155.49 Bn for Sports Betting Market Size 2019 Growing at 8.83% CAGR Through 2024. Globe Newswire. [online] Available at: https://www.globenewswire.com/news-release/2019/07/26/1892289/0/en/Research-US-155-49-Bn-for-Sports-Betting-Market-Size-2019-Growing-at-8-83-CAGR-Through-2024.html. Accessed: 9 December 2020.
3. M. Beckler, H. Wang, and M. Papamichael, (2009). NBA Oracle. Project report for machine learning course at Carnegie Mellon University. https://www.mbeckler.org/coursework/2008-2009/10701_report.pdf
4. J. Uudmae, (2017). CS229 Final Project: Predicting NBA Game Outcomes. http://cs229.stanford.edu/proj2017/final-reports/5231214.pdf
5. Ahmed, T., Oreshkin, B. and Coates, M., (2007). Machine learning approaches to network anomaly detection. In Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques (pp. 1-6). USENIX Association.
6. Omar, S., Ngadi, A. and Jebur, H.H., (2013). Machine learning techniques for anomaly detection: an overview. International Journal of Computer Applications, 79.
7. Basketball Reference (2020). Basketball Statistics and History. [online] Available at: https://www.basketball-reference.com/. Accessed: 1 December 2020.
8. En.wikipedia.org. (2020). Efficiency (Basketball). [online] Available at: https://en.wikipedia.org/wiki/Efficiency_(basketball). Accessed: 9 December 2020.