

# Prediciting the Premier League results using Machine Learning

Tariq Ahmed  
161134893  
Thomas Roelleke  
MSC Electronic Engineering Big Data  
Science

**Abstract**—This project aims to analyse two extensive datasets, involving the statistics of multiple footballing seasons and the player ratings provided by EA yearly. The project aims to evaluate the features used, to observe the importance of EA's Ratings for predicting results. In addition, the project aims to predict the Premier League matches results, by predicting one as a win, two as a draw and zero as a loss for the home team. Similarly, the project intends to predict the league standing of the 2019/2020 footballing season, and then compare the results obtained to the real standing. Additionally, this project aims to observe if the models trained are also suitable for predicting the home scores as well. Finally, using multiple machine learning models in python, this project hopes to identify which technique is most suited for such a task.

## 1.INTRODUCTION

Prediction and forecasting have been topics that have lasted for centuries. From societies such as ancient Greece where people would go to oracles for prophecies regarding the future, to men like Nostradamus's predicting major events in the 16<sup>th</sup> century, to today where we use algorithms to predict the weather and economic growth. An area which has seen the application of algorithms for prediction is sport. Bookmakers will use information regarding teams to apply different odds on the outcome. Information such as player's form, last 5 game outcomes, shots on target and more are all vital factors when predicting the result of a match.

Using a variety of models, this report therefore aims to provide how important certain factors like EA's independent rating system, are at predicting the results of games and a team's final position in the league standing. This will then be used to evaluate how effective both the models and features are when predicting the outcome of a season.

### 1.1 Related Research:

There have been several projects trying to predict the outcome of a Premier League season with similar accuracy. A recent project report was authored in 2018 by Corentin Herbinet from Imperial college London [1]. The project took a dataset from Kaggle called the "Kaggle European Soccer Database". The study then filtered the dataset down to 2014/2015 and 2015/2016 seasons containing 3,800 matches from the English Premier League, French Ligue 1, German Bundesliga, Spanish Liga, and Italian Series A. They analyzed 88,340 shots with over 100 different teams. The data used, had factors such as League ID, Season, home team goals, away team goals, home team possession, away team possession, home win odds, draw odds and away odds.

The study then generated expected goals (xG), match xG, ELO ratings calculated using expected goals value and actual match performances to then generate a prediction of the result

using different models. In generating the xG, the models used were Logistic Regression, KNN, Gaussian Naïve Bayes and Random Forest. These models all preformed in similar accuracy with 50.75%,50.45%,50.5% and 50.5% respectively. The greatest results gathered were a predictive accuracy of 51.1% and F1 score of 38.2%. The study also concluded that using Gaussian Naïve Bayes was the best model to predict the xG used to generate the result.

### 1.2 The dataset:

The datasets in this report are based on the Premier League season from 2014 to 2019 and the EA player dataset from 2015 to 2020. The EA's player dataset was obtained from Kaggle while the Premier League season dataset was provided by football-data.co.uk. As such there are no ethical or privacy issues in using the dataset for this project as both datasets are intended for public use. Both sets of databases use a CSV format.

Historical Premier League Data:

Variable Name	Description
Home Team	This represents the home team playing in the match
Away Team	This represents the away team playing in the match
HTHG	This represents half time home team goals in the match played
FTAG	This represents full time away team goals in the match played
HS	This represents home team shots in the match played
AS	This represents away team shots in the match played
HST	This represents home team shots on target in the match played
AST	This represents away team shots on target in the match played
FTR	This represents full time results in the match played

EA player rating:

Variable Name	Description
Year	This represents the year the game was released
Club	This represents the team in the Premier League
Overall	This represents the overall rating that EA have given to the player
Team position	This represents the position the player plays within the team
Pace	This represents the attribute speed rating that EA given to the player

Shooting	This represents the attribute shooting rating that EA given to the player
Passing	This represents the attribute passing rating that EA given to the player
Dribbling	This represents the attribute dribbling rating that EA given to the player
Defending	This represents the attribute defending rating that EA given to the player
Physic	This represents the physical attribute rating that EA given to the player

## 2. BACKGROUND

### 2.1 Linear Regression:

Knowledge of Linear Regression is important, as it sets up the foundation for understanding Regression and how it relates to Logistic Regression. Linear Regression is a statistical method, that attempts to model the relationship between two variables, by fitting a linear equation to the observed data. It is not necessary that one variable causes the other, but there is some association between the two variables.

Given a data set:

$$\left[ y_{i_1}, x_{i_1}, \dots, x_{i_p} \right]_{i=1}^n \quad (2.1)$$

A Linear Regression model assumes that there is a linear relationship between the dependent variable  $y$  and the independent variable  $x$ . The relationship is modelled using an error term  $\epsilon$ , which acts as an undetected random variable that adds “noise” to the linear relationship. The model below is a representation of its form.[2]:

$$y_i = \beta_0 + \beta_1 x_i + \dots + \beta_{1p} x_{ip} + \epsilon_i \quad i = 1, \dots, n \quad (2.2)$$

The figure shown below is a visualization of Linear Regression. The data points shown in black are observed data which have been manipulated due to random changes (error term) in red. The line in blue represents the linear underlying relationship between dependent variable  $y$  and independent variable  $x$ .

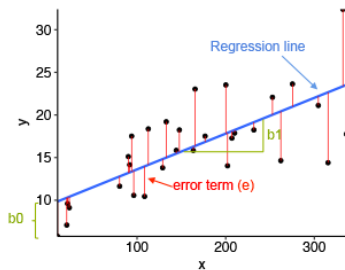


Figure 2.1: Visualization of a 2-variable linear regression [3]

### 2.2 Logistic Regression:

While Linear Regression is beneficial for continuous data, it is not a suitable technique when the variables are binary. Linear models create inadequacies, when fitted to a binary

response. As such, variables are modelled using probability of the response, by a function giving an output between 0 and 1. This type of functions is Logistical. This can be achieved through many different functions, but the most common function is Sigmoid function.

$$P(x) = \frac{1}{1+e^{-x}} \quad (2.3)$$

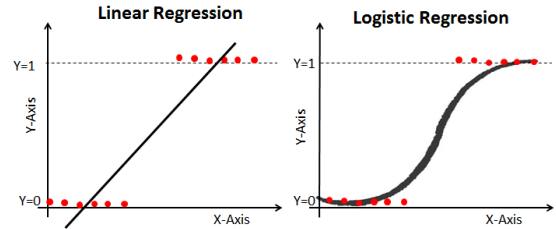


Figure 2.2 and 2.3: showing Linear and Logistic Regression model fit on binary variables [4]

The equation in 2.3 can be used to predict a binary response for multiple predictors.

$$P(X) = \frac{e^{\beta_0 + \beta_1 X + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X + \dots + \beta_p X_p}} \quad (2.4)$$

where  $\beta_p$  represents the gradient coefficients, much like in linear regression. The unknown parameters are found using maximum likelihood estimation. This is applied for fitting in a Logistic Regression model. Maximum likelihood estimation works by trying to gain the  $\beta$  parameters. This is done by finding estimates, so the predicted probability for each observation resembles the observed status as close as possible. The equation below uses the maximum likelihood to form the function [4].

$$l(\beta_0, \beta_1) = \prod_{i:y=1} P(X_i) \prod_{i':y'=1} [1 - P(x'_{i'})] \quad (2.5)$$

### 2.3 Decision Tree:

Decision Trees is a supervised machine learning technique that partitions data into subsets. A binary split starts the partitioning process which continues until it can no longer split the data. Decision Tree's purpose is to condense the training data into the smallest tree possible [5]. A Decision Tree is created by two types of components, nodes, and branches. On each node, a feature from the data is assessed if it can split the observations in the training process; or if it should make a specific data point to follow a certain path when making a prediction. Decision Trees are constructed recursively, by assessing different features, and by using the feature that best partitions the data at each node [6].

The root node is at the top of the tree, it is used to evaluate the feature that best splits the data. Next is the intermediate nodes, which evaluate the features. Finally, there are the leaf nodes which predicts categories or numerical values.

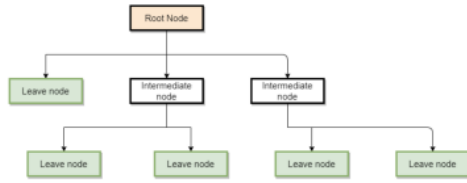


Figure 2.6: Decision Tree [11]

Decision Trees are trained by evaluating some metrics, like the Gini index, the Entropy for categorical decision trees, or the Residual or Mean Squared Error for regression trees. The process of evaluation is different depending on if the nodes are continuous or discrete. The process for discrete features is that all values are evaluated, creating N calculated metrics for each variable, where N is the number of possible values for each categorical value. Whereas, for continuous features, the mean of each two-consecutive value of the training data are used as possible thresholds.

The results are processed using a calculated metric (Gini or MSE) for each variable/threshold. Next, the variable/threshold combination that provides the largest/smallest value for a specific metric is picked.

Decision Trees can be created in many ways, but the most common method is the classification and regression tree (CART) algorithm proposed by Breiman(1984). Decision Trees split the training data into homogenous subgroups and then fit a simple constant to each subgroup. The subgroups are formed recursively, using binary partitions like providing yes/no question to each feature. This is completed multiple times and only stops when a suitable criterion is met (e.g., the maximum depth of the tree is reached). When partitioning is completed, the model predicts the output based on the class that represents the majority. Predicted probabilities can be found using the proportion of each class within the subgroup.

The aim of each node is to obtain the best features, and to split the remaining data into two areas (R1 and R2) to minimize the overall error between the actual response (variable  $y_i$ ) and the predicted constant ( $c_i$ ). The Sum of Squared Error function (SSE) is shown by the below equation [7].

$$SSE = \sum_{i \in R_1} (y_1 - c_1)^2 + \sum_{i \in R_2} (y_1 - c_2)^2 \quad (2.6)$$

### 2.3.1 Tree pruning:

Pruning is important to obtain the optimal subtree. The optimal subtree is found using a cost complexity parameter known as  $\alpha$ . If the cost of adding another variable to the terminal nodes of tree T is greater than value of  $\alpha$ , then tree building stops. This is achieved using the equation below.

$$\text{minimise } \{SSE + \alpha|T|\} \quad (2.7)$$

### 2.4 Random Forrest:

A Random Forest is made up of many single Decision Trees that operate together. Each tree in Random Forest gives a classification, the class with the most “votes” by the trees becomes the model’s prediction. Decision Trees are more likely to overfit, due to trees having many branches. Random Forest is less likely to have issues with overfitting compared

to Decision Tree; as Random Forest average the votes of many decision trees to restrict overfitting as well as reduce error due to bias.[8]

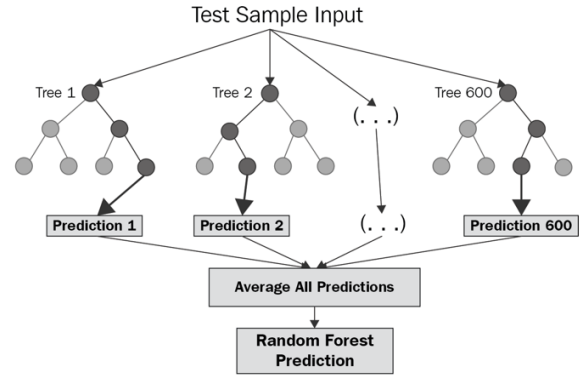


Figure 2.7: Random Forests [8]

### 2.5 Support Vector Classification:

Support vector machines is a machine learning model that finds a hyperplane in an N-dimensional space (N number of features) that clearly classifies the datapoints. Hyperplanes are decision boundaries, used to classify the datapoints depending on which side of the hyperplane the datapoint falls on. The dimension of the hyperplane is determined based on the number of features provided. Such that if the input features are 2, then the hyperplane is a line while if the input features are 3 then the hyperplane is a 2-dimensional plane.

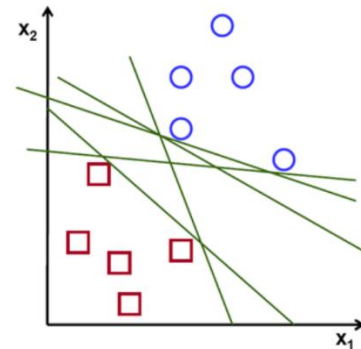


Figure 2.8: SVM Hyperplane for classification [9]

Support vectors are datapoints which are near to the hyperplane, which are used to control the position and orientation of the hyperplane. Using support vectors, we aim to maximize the margin of the classifier. Using a linear function, based on if the output is greater than 1, the datapoint is classified into one class. Whereas if the output is -1 then it is classified into the other class.

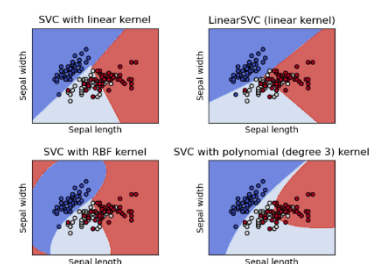


Figure 2.9: SVM with different kernel tricks [10]

When the data is not linearly separable, a kernel is used such as Radial Basis Functions (RBF) or polynomial functions, to map the data into high-dimensional feature spaces and find a suitable high-dimensional hyperplane.

## 2.6 Naive Bayes classifier:

A Naive Bayes classifier is a probabilistic machine learning model used for classification. It can predict the probability distribution over its classes given its input variables. It is named naïve as it assumes that all features are independent which allows the use of bayes theorem:[11]

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \quad (2.8)$$

The variable H is the class variable, the outcome based on its condition. Variable E represent the features.

Given E:

$$E = (x_1, x_2, x_3, \dots, x_n) \quad (2.9)$$

As such replacing X and expanding using the chain rule we achieve:

$$P(y|x_1, \dots, x_n) = \frac{p_1(x_1|y)P(x_2|y) \dots P(x_n|y)P(y)}{P(x_1)P(x_2) \dots P(x_n)} \quad (2.10)$$

Using the independence assumption, this leads to the following:

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y) \quad (2.11)$$

MAP (Maximum A Posteriori) estimation enables the formation of a rule to select the class that return the output by choosing the most probable hypothesis.

When the predictors have continuous values, it is assumed that the values are sampled from a gaussian distribution.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)} \quad (2.12)$$

The parameters are estimated using Maximum Likelihood Estimation.

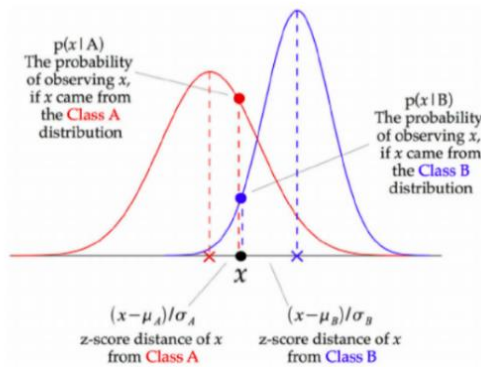


Figure 2.10: Gaussian Naive Bayes method [12]

## 2.7 Model performance:

It is important to evaluate accuracy when trying to find which model is most suitable for prediction. Therefore, several

metrics will be used when analyzing the performance of each model. This is because one metric may strongly favour one model but may not indicate it the best overall model for prediction in all categories.

### 2.7.1 Classification Accuracy:

Accuracy can be simply calculated using the equation below:

$$Accuracy = \frac{\text{total correct prediction}}{\text{total predictions made}} \quad (2.13)$$

However, this metric is somewhat limited and misleading, in determining the accuracy depending on if the test data is imbalanced. For instance, if the dataset has 99% of its sample belonging to class A and 1% of samples belong to class B. It would indicate a strong accuracy which may not be correct.

### 2.7.2 Confusion Matrix:

The confusion matrix is a visual representation of the performance in matrix form.

- True Positives: Where the predicted output is TRUE, and the actual output is TRUE
- True Negatives: Where predicted output is FALSE, and the actual output is FALSE
- False Positives: Where the predicted output is TRUE, and the actual output is FALSE
- False Negatives: Where the predicted output is FALSE, and the actual output is TRUE

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.11: Confusion Matrix [13]

From this several metrics can then be calculated. Which are:

$$Recall = \frac{TP}{TP + FN} \quad (2.14)$$

$$Precision = \frac{TP}{TP + FP} \quad (2.15)$$

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (2.17)$$

## 3. METHODOLOGY

### 3.1 Import the dataset:

The historical data was split per season, into a csv file where each dataset contained 380 matches, using pandas within python the seasons 2014/15 to 2019/20 were imported separately. In pandas the five seasons were then concatenated together. To match the years of the historical data, the same EA's player rating years were used. Each EA's player rating dataset contained 15466 entries. However as only the Premier League players were needed for this report, players were filtered out by the clubs competing in the 2019/2020 season. The first step was to clean and pre-process the data.

### 3.2 Data preprocessing and cleaning:

An important step before building the model is to examine and pre-process the data to ensure it is compatible for training and testing the models.

The preprocessing steps were completed as shown below:

- After importing the dataset, the Categorical data (Club names) were encoded using a label encoder, which was needed so the models could predict. The Club names were stored in a dictionary.
- Both datasets were then merged using the “Season” column, which was created to store the season the match was played in. And the “Release year” column, which was created to store the year the ratings were published in. The home team and away team ratings were created by merging on the home and away encoded labels.
- Dates needed to be formatted correctly, as the historical data was from <https://football-data.co.uk/> which used a day month year format. This was corrected by using a method in pandas with the parameter “day first”.
- It was found that the Club Sheffield United was not in the historical data. The club had not played in the Premier League since 2006 and finally gained promotion in 2019. As 2019 was the year being predicted, Sheffield United was replaced with Stoke who are a similar level to Sheffield United.
- Using moving averages, explained in the next section, some data elements contained nulls. This was because there was no imported data before 2014, and so the nulls were filled with zeros to ensure the models predicted results.
- Only the starting 11 players were used, and so reserves and substitution players were removed. This reduced the number of players in the EA’s datasets to 220 per year. After filtering the players in each data frame, some specific ratings were selected as features to be used.
- Finally, the dataset was split using a function that splits the data into train and test subsets. This was used to tune the models before predicting the results of the 2019/2020 season.

### 3.3 Feature selection and engineering:

From the historical dataset the features selected were the Date, Home Team, Away Team, the full-time team goals, the halftime team goals, halftime team shots on target, full-time team shots on target, team corners and halftime and full-time results. These features were stored for both the home and away team respectively.

To predict the matches, the historical data could not directly be used, as this data is unknown when predicting a game to be played. As such a moving average of the features selected above were used. This was completed, by selecting the date and the home team as the variables to iterate through. By iterating through the dates, and checking the iterable of the home team, we could select the correct rows and find the average. By iterating through the next date with the same home team, we could create a moving average.

This is based on the general equation 2:

$$\bar{asm} = \frac{1}{M} \sum_{i=0}^{n-1} x_{M-i} \quad (3.1)$$

A running total was also used to find the overall form of a team, using the points gained in the last 1,3 and 5 matches. By assigning the results of a win as 1, draw as 2 and loss as 0 for the home team. And then calculating the points gained, we could sum up the points gained over 1,3 and 5 matches.

$$y = \sum_{i=0}^{n-1} x_M \quad (3.2)$$

As stated in the introduction, one aim of the project was to evaluate if EA’s player rating would influence the predictions of results. To carry out this aim, specific ratings were extracted from the EA’s player rating datasets.

These ratings were split based on player’s positions. Such that if the player was defensive minded, specific ratings were taken i.e., physical and defending. If the player was attacking minded, the ratings chosen were pace, shooting, passing, and dribbling. This was done as an indication of how strong a team was at defending and attacking. Each player in the starting 11, overall player rating was averaged together and became the team’s overall rating. Using the equation 1. The difference between the home team and away team rating was used in the model.

$$R_i = R_i^H - R_i^A \quad (3.3)$$

PCA was applied to the X features, to reduce dimensionality within the dataset. This is achieved by transforming a set of variables into a smaller number of variables that contain information of the larger set. While this has a slight negative affect on accuracy, it is done to simplify the data. Making it easier to visualize the data and lower the computation time. After scaling the X variables, X variable were then split between X train and X test.[14]

- ❖ Feature selection – The feature selection employed in this project is exclusive to this report. The different techniques used to filter ratings and aggregate the rating chosen are specific to this report. For example, specifically storing ratings of players in the starting 11. And then further splitting ratings based on what rating is important to that position in the field. For instance, generally the defensive rating is more important to a centre back then shooting rating. Therefore, the only rating a centre back influenced was the defending and physical rating.
- ❖ Missing data - Not having Sheffield United in the historical data was a big challenge when predicting the 2019/2020 season. The model could not predict without 20 exact teams, which make up a premier league table. As such removing Sheffield United without replacing them would lead to an error. Replacing Sheffield united with Stoke is an original idea that was chosen as Stoke are of similar standing to Sheffield United. Stoke were chosen as they had appeared in the football-data dataset and had comparable stats and similar type of defensive play.

### 3.4 Applying Logistic Regression - optimum parameters:

Logistic regression was applied to the data. The function “LogisticRegression()” in python is called as well as the library “GridSearch”. The “LogisticRegression()” function uses the maximum likelihood estimation, explained in the background section 2.2. Then all the possible values for the parameters are then provided. In Logistic regression the hyperparameters that are provided to be tuned were the “C”, “penalty”, and “solver”. The solver are different methods that minimize the cost function of the equation. The different solvers used were 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'. The C parameter is inverse regularization strength, it is used as the inverse of the regularization penalty to increase the magnitude of the parameter values. C was set as 100, 25, 10, 1, 0.1 and 0.01. The penalty is used to specify the size of the vector which is used in penalization. The penalty is set to l1 or l2 based on the solver used.

Using “GridSearch”, a comprehensive search is completed within python over the specified parameter values. This is done by building a model of each possible combinations of hyperparameters supplied, which then evaluates each model combination. Then it returns the best parameters to use on the data. Afterwards, fitting the training data with “GridSearch” and printing the “best estimator”, we can then gain the optimal hyperparameters for predicting on the testing data. This was done for all the models used within this project. However, GridSearch suffers due to dimensionality. This is because as we evaluate the model based on its hyperparameters, the number of parameters grows exponentially which increases computation time. In addition, using GridSearch does not guaranteed the best hyperparameters are found. After using GridSearch and printing the best estimator. The optimal parameters were found and set as C=0.1, penalty=l1 and solver=saga.

### 3.5 Applying Decision Trees- optimum parameters:

Another model used was Decision Tree. This was achieved by using the “DecisionTreeClassifier()” function in python. This is completed using binary recursive partitioning explained in section 2.3. The parameters chosen to be optimized in Decision Tree were “max depth”, “random state”, “min sample split”, “min sample leaf” and “criterion”. “Min samples split” is the minimum number of samples needed to split a node, and it was set between a range of 1 to 10. “Min samples leaf” is the method of selecting samples for training each tree, and it was set between a range of 1 to 6. The “max depth” is the maximum number of levels within the tree, and it was set between a range of 1 to 12. “Criterion” is the function used to measure the quality of the split, both “gini” and “entropy” were explained in section 2.3. Random state is the parameter controlling the randomness of the estimator. This was set as a value between 0 to 50. After using GridSearch and printing the best estimator. The optimal parameters were found and set as criterion=entropy, max depth=3, min samples leaf=1, min samples split=2 and random state=0

### 3.6 Applying Random Forest - optimum parameters:

Random Forest was also used in this project. This was called using the “RandomForestClassifier()” function in python. This is completed using binary recursive partitioning explained in section 2.4. The parameters chosen to be

optimized in Random Forest were “N estimators”, “max features”, “max depth”, “min sample splits” and “min sample leaf” and bootstrap. “N estimators” is chosen to decide how many trees will be used in Random Forest. This hyperparameter was given a set of values 100, 300, 500, 800 and 1200. The “max features” is used to obtain the number of features to consider when looking for the best split. It was set to choose between “auto” and “log2”. Auto uses the square root of the number of features, whereas log2 uses the log2 of the number of features. The “max depth”, “min samples split” and “min samples leaf” all have the same definition as above in section 3.5. “Max depth” values given were 5, 8, 15, 25 and 30. “Min samples split” was set as 2, 5, 10, 15 and 100. “Min samples leaf” was set to 1, 2, 5 and 10. After using GridSearch and printing the best estimator. The optimal parameters were found and set as random state = 1, max depth = 16, bootstrap=True, max features=auto, n estimators = 1200, min samples split = 2, min samples leaf = 1.

### 3.7 Applying SVC - optimum parameters:

A different technique also applied was support vector classification. The function “SVC()” was called in python. The “SVC()” function which uses the boundaries and hyperplanes explained in the background section 2.5. The parameters chosen to optimize the SVC were “C”, “Gamma” and “kernel”. The kernel is also explained in background section 2.5. It is used to map the data in different high-dimensional feature spaces. The kernel used were “rbf”, “poly” and “sigmoid”. Gamma is a parameter for nonlinear hyperplanes. The higher the gamma value the more the influence a single training example has, meaning that other examples must be closer to be affected. The Gamma was set as 0.0001, 0.001 and 1. The C parameter is inverse regularization strength where it trades off misclassification of training examples against simplicity of the decision surface. This value was set as 1, 10, 100 and 1000. After using GridSearch and printing the best estimator. The optimal parameters were found and set as C=10, gamma=0.01, kernel=rbf.

### 3.8 Applying Gaussian Naïve Bayes - optimum parameters:

The final technique applied to the data was Gaussian Naïve Bayes. The function “GaussianNB()” was called with grid search. The “GaussianNB()” function uses the maximum a posteriori (map) estimation explained in the background section 2.6. “Var smoothing” is the only parameter tested using “GridSearch”. “Var smoothing” is the portion of the largest variance of all features, which is added to variance to calculate stability. The values set were between 0 and 1. After using GridSearch and printing the best estimator. The optimal parameter was found and var smoothing was set to 0.8111308307896871.

❖ Due to how some features were created, the values needed to optimise the hyperparameters are exclusive to this project. This is because how data is manipulated in this project is unique. For example, creating features based on player position, is a different angle to other project using EA’s ratings. As such both the useful data and noise for the models is exclusive to this project.



#### 4.EVALUATION

To summarize the tuning applied in section 3. The individual model's performance after tuning was compared. The prediction kept changing because of the function "train test split". To gain repeatable results, the train-test split function was modified so that the "random state" was set to an integer. This fixes the pseudo-random number generator used to split the dataset. A breakdown was shown below using weighted average of Precision, Recall, F1-score, and the Accuracy.

Model	Precision	Recall	F1-score	Accuracy
Gaussian Naive Bayes	0.49	0.53	0.48	0.53
Logistic Regression	0.50	0.54	0.48	0.54
Decision Trees	0.47	0.51	0.48	0.51
Random Forest	0.48	0.53	0.45	0.53
SVC	0.45	0.48	0.46	0.48

The table confirms that Logistic regression was the best performing model. It achieved amongst the highest scores across all categories. With the F1-score being the most important metric, with a score of 0.48 and the highest accuracy of 0.54, it was the best model suited for the task. F1 score is the most important metric because results are likely to be imbalanced with certain teams being better performing than others. In F1 score, Random Forest was the weakest performing. Unexpectedly the Decision Trees model outperformed the Random Forests model. This contradicts the theory that Random Forests should perform better than Decision Trees. This is said because it is a stronger modelling technique which is more robust than using a single Decision Tree. As Random Forest aggregates many Decision Trees to restrict overfitting. This may be caused by noise in the data. As stated in section 3.6, the number of trees was set to 1200. Through analysis of the error rate, it appeared that 1200 was an adequate number as the error rate seemed to stabilize. Therefore, suggesting that additional trees may be needed to provide a true indication of minimal error rate. Based on the table above Logistic regression was chosen to predict the final 2019/2020 season standing.

```
[('Liverpool', 90.0),
 ('Man City', 90.0),
 ('Leicester', 87.0),
 ('Bournemouth', 81.0),
 ('Brighton', 75.0),
 ('Newcastle', 75.0),
 ('Arsenal', 69.0),
 ('Man United', 56.0),
 ('Watford', 56.0),
 ('Everton', 53.0),
 ('Aston Villa', 51.0),
 ('Crystal Palace', 47.0),
 ('Tottenham', 45.0),
 ('West Ham', 41.0),
 ('Chelsea', 38.0),
 ('Wolves', 35.0),
 ('Norwich', 34.0),
 ('Burnley', 33.0),
 ('Stoke', 29.0),
 ('Southampton', 28.0)]
```

Figure 4.1: Table prediction using Logistic regression

The predictions were much weaker when predicting against the 2019/2020 season, when the historical data was not supplied. To predict the results, the latest moving averages from the 2018/2019 season were used. By combining the latest home and away stats from the previous season. A csv file was created as the input for predicting the games. By looking at Figures 4.1 and the Premier League table in the appendix, we can see that the first two positions were correct while the points were not. Furthermore, the points spread does seem to match that of the Premier League table even though most of the club positions were incorrect.

To understand what features were the most influential in predictive power, we must evaluate the ranks of each feature. As Logistic regression was best F1 scoring model, its features was evaluated and ranked.

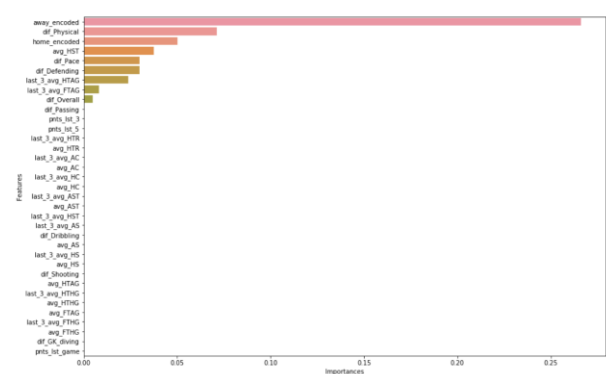


Figure 4.2: Bar plot comparing the different features importance for random forest

As shown in the figure above, the away team was the most influential feature. This should be expected, as depending on who the home team is playing will have the greatest effect on the outcome of the game. If the away team is very defensive orientated, the result is more likely to be a draw than playing against another attacking minded team. The difference in EA rating in Physicality, Pace and Defending ratings were also influential variables. Therefore, we can imply that EA ratings are a somewhat acceptable way of judging a team's capabilities. These features maybe expected to be influential, as each feature is a representative of how a team plays. For example, if the away team has a high pace rating, they are more likely to play down the wings and play better on the counter. Whereas, if the home team has a high defense rating and physicality, they are more likely to be defensive minded and play for a draw. Another influential variable was the average home team shots on target. This may be expected as it is an indication of how attacking minded the home team is. If the home team has many shots on target in a game, they are more likely to score a goal and win the game. Most other factors like the points gained in the last game, and other differences in EA ratings were proven to not be influential.

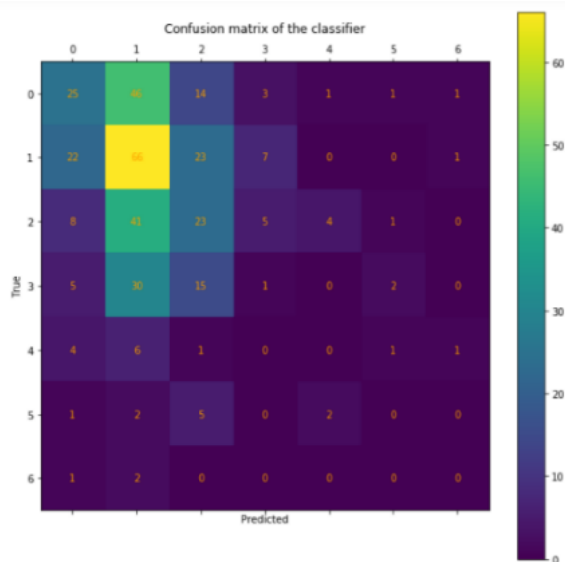


Figure 4.3: Confusion matrix of full-time home goals

After predicting the results of a football season, a tuned Logistic regression model was used to predict the full-time home goal scored in the dataset. Looking at figure 7.13 in the appendix, the F1-score was low at 0.33, as well as the accuracy at 0.37. However, using Figure 4.3 for comparison, we can see full-time home goals predicted where much closer to true results than first shown. Based on the matrix table we can see that most full-time home results had a value between 0 and 2. With a true value of 0 goals scored being 91, 25 predictions were correct, and 46 predictions were 1 goal away from the true results. Looking at 1 goal as the true result, we can see that the home team scoring 1 goal by the full-time was 119, 66 predictions were correct, and 45 predictions were 1 goal away from the true results. The split between 2 goals predicted and 0 goals predicted were even at 22 and 23, respectively. The testing data did not contain a lot of games where the home team scored more than 2 goals and so predictions was much worse. Maybe this was due to the matches being 90 minutes long and so there is no extra time for more than 2 goals being scored. We can see from the matrix that model tended to predict one less goal scored than real results. This likely due to tuning applied to the model meaning the model tends to underfit.

- ❖ Goal comparison - The comparison of goal scored using the confusion matrix is unique to this project. By checking the goals predicted against the actual goals, we get a better insight in how the model was at predicting. Compared to using standard metrics like F1 and accuracy.

## 5.CONCLUSION

This study revolved around the use of Logistic Regression, Gaussian Naïve bayes, Decision Trees, Random Forest, and SVC to predict the results of matches. This study aimed to explore which factors, specifically EA's ratings were effective at predicting the result of a game. By predicting the 2019/2020 season with logistic regression. The precision achieved was 0.37 and the accuracy was 0.36. While the precision may not seem high, it also needs to be considered that the 2019/2020 season was chosen specifically. This is because it was a season of "unpredictableness" as it was affected by COVID-19. This meant fans could not enter the

stadium to support their teams. This also meant that home advantage was not relevant, whereas in previous seasons fans could influence the game. In addition, the features used were supplied by selecting the moving averages of the previous season 2018/2019. As such the results were much worse, compared to supplying the models with the data from the current season.

Player ratings were found to be somewhat influential when predicting the results of the games. Only a few of EA's rating features had any influence on the result, which were "dif\_Pace", "dif\_Defending" and "dif\_Overall". The most important EA's rating was the "dif\_Pace" with "dif\_Defending" being ranked one position lower in importance. This shows that using EA Rating did have a positive impact on predicting results. The difference in overall rating was not as important as first believed, as it was ranked lowest in importance on the feature importance chart.

To an extent, this project did attain its aim in understanding which variables influence the result of a football game. The study was able to match similar scores to those gathered from previous projects while using different features to predict the results. To conclude, prediction of a football match is a complex challenge. This is because there are many factors that can influence the results. Some of these factors cannot be computed and other factors are simply down to chance. It is also important to consider that the sample of data taken for analysis was only 2014/2015 to 2018/2019 seasons, while these years matched those of the EA player rating, supplying more data can increase performance. As a result of the period chosen, clubs like Sheffield United could not be predicted as there was no data of them playing in the league.

As mentioned throughout the project, originality is driven by different angles used to tackle challenges within this report. How feature selection is done in this project is original. The angle taken was to precisely find what is deemed as important factors and then remove the excess which is likely to be noise. Such as, rating features are specifically calculated based on position. Also, the Sheffield United issue is a specific problem that occurs when trying to predict the 2019/2020 season. Finally, the report also looks at alternative methods for evaluating models instead of traditional metrics such as F1 and accuracy.

As with any study, improvements can be made, and extra things can be considered for future development. For example, the addition of more datasets such as the Spanish or German league could have led to a more accurate and confident model. However, what can also occur when increasing the amount of data is worse performance. As more noise is added to the data. Exploration of Deep Learning models like the Convolutional Neural Networks or Recurrent Neural Networks could lead to more effective models. However, as there is no context, it is difficult to explain why certain outputs were predicted, as Deep learning has a "black box" nature.



## 6. 6 BIBLIOGRAPHY:

- [1] HERBINET, C., 2018. *Predicting Football Results Using Machine Learning Techniques*. postgraduate. Imperial College London.
- [2] Greenwell, B., 2020. Chapter 4 Linear Regression | Hands-On Machine Learning With R. [online] Bradleyboehmke.github.io. Available at: <https://bradleyboehmke.github.io/HOML/linear-regression.html>
- [3] Kassambara, A., 2018. *Simple Linear Regression in R*. [online] Sthda.com. Available at: <http://www.sthda.com/english/articles/40-regression-analysis/167-simple-linear-regression-in-r/> [Accessed 28 June 2021].
- [4] Navlani, A., 2019. *Understanding Logistic Regression in Python*. [online] datacamp.com. Available at: <https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python> [Accessed 28 June 2021].
- [5] Webfocusinfocenter.informationbuilders.com. 2020. Explanation Of The Decision Tree Model. [online] Available at: [https://webfocusinfocenter.informationbuilders.com/wfapen/TLs/TL\\_rstat/source/DecisionTree47.htm](https://webfocusinfocenter.informationbuilders.com/wfapen/TLs/TL_rstat/source/DecisionTree47.htm) [Accessed 28 June 2021].
- [6\*] Zornoza, J., 2020. Decision Trees Explained. [online] Medium. Available at: <https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6> [Accessed 28 June 2021].
- [7\*] Greenwell, B., 2020. Chapter 9 Decision Trees | Hands-On Machine Learning With R. [online] Bradleyboehmke.github.io. Available at: <https://bradleyboehmke.github.io/HOML/DT.html> [Accessed 28 June 2021]
- [8] Chakure, A., 2019. *Random Forest Regression*. [online] medium.com. Available at: <https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f> [Accessed 28 June 2021].
- [9] Gandhi, R., 2018. *Support Vector Machine — Introduction to Machine Learning Algorithms*. [online] towardsdatascience.com/. Available at: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> [Accessed 28 June 2021].
- [10] Sklearn, n.d. *1.4. Support Vector Machines — scikit-learn 0.24.2 documentation*. [online] Scikit-learn.org. Available at: <https://scikit-learn.org/stable/modules/svm.html> [Accessed 28 June 2021].
- [11] Brownlee, J., 2019. *A Gentle Introduction to Bayes Theorem for Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/bayes-theorem-for-machine-learning/> [Accessed 28 June 2021].
- [12] Majumder, P., n.d. *Gaussian Naive Bayes*. [online] OpenGenus IQ: Computing Expertise & Legacy. Available at: <https://iq.opengenus.org/gaussian-naive-bayes/> [Accessed 28 June 2021].
- [13] Sarang Narkhede (2018). Understanding Confusion Matrix. [online] Medium. Available at: <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62/> [Accessed 28 June 2021].

- [14] Jaadi, Z., 2021. *A Step-by-Step Explanation of Principal Component Analysis (PCA)*. [online] Built In. Available at: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis> [Accessed 30 June 2021].
- [15] Premierleague, 2019. *Premier League Table, Form Guide & Season Archives*. [online] Premierleague.com. Available at: <https://www.premierleague.com/tables?co=1&se=274&ha=-1> [Accessed 28 June 2021].

## 7. APPENDIX:

```
GaussianNB(var_smoothing=0.8111308307896871)
```

	precision	recall	f1-score	support
0	0.53	0.42	0.47	106
1	0.55	0.87	0.67	182
2	0.00	0.00	0.00	83
accuracy			0.54	371
macro avg	0.36	0.43	0.38	371
weighted avg	0.42	0.54	0.46	371

Figure 7.1 classification report for Naïve Bayes before tuning

	precision	recall	f1-score	support
0	0.44	0.53	0.48	106
1	0.58	0.74	0.65	182
2	0.36	0.06	0.10	83
accuracy			0.53	371
macro avg	0.46	0.44	0.41	371
weighted avg	0.49	0.53	0.48	371

Figure 7.2 classification report for Naïve Bayes after tuning

```
LogisticRegression(C=0.01, penalty='l1', solver='saga')
```

	precision	recall	f1-score	support
0	0.48	0.47	0.47	106
1	0.55	0.80	0.65	182
2	0.00	0.00	0.00	83
accuracy			0.53	371
macro avg	0.34	0.42	0.37	371
weighted avg	0.40	0.53	0.45	371

Figure 7.3 classification report for Logistic Regression before tuning

	precision	recall	f1-score	support
0	0.48	0.56	0.52	106
1	0.58	0.77	0.67	182
2	0.33	0.02	0.04	83
accuracy			0.54	371
macro avg	0.47	0.45	0.41	371
weighted avg	0.50	0.54	0.48	371

Figure 7.4 classification report for Logistic Regression after tuning

```
DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=0)
```

	precision	recall	f1-score	support
0	0.48	0.44	0.46	106
1	0.54	0.81	0.65	182
2	0.00	0.00	0.00	83
accuracy			0.53	371
macro avg	0.34	0.42	0.37	371
weighted avg	0.40	0.53	0.45	371

Figure 7.5 classification report for Decision Trees before tuning

	precision	recall	f1-score	support
0	0.49	0.37	0.42	106
1	0.56	0.75	0.64	182
2	0.27	0.16	0.20	83
accuracy			0.51	371
macro avg	0.44	0.42	0.42	371
weighted avg	0.47	0.51	0.48	371

Figure 7.6 classification report for Decision Trees after tuning

```
RandomForestClassifier(bootstrap=False, max_depth=30, min_samples_leaf=10)
```

	precision	recall	f1-score	support
0	0.45	0.37	0.40	106
1	0.55	0.82	0.66	182
2	0.30	0.04	0.06	83
accuracy			0.52	371
macro avg	0.43	0.41	0.38	371
weighted avg	0.46	0.52	0.45	371

Figure 7.7 classification report for Random Forest before tuning

	precision	recall	f1-score	support
0	0.49	0.39	0.43	106
1	0.54	0.84	0.66	182
2	0.33	0.01	0.02	83
accuracy			0.53	371
macro avg	0.45	0.41	0.37	371
weighted avg	0.48	0.53	0.45	371

Figure 7.8 classification report for Random Forest after tuning

```
SVC(C=1, gamma=0.001, kernel='sigmoid')
```

	precision	recall	f1-score	support
0	0.48	0.51	0.50	106
1	0.57	0.81	0.67	182
2	0.00	0.00	0.00	83
accuracy			0.54	371
macro avg	0.35	0.44	0.39	371
weighted avg	0.42	0.54	0.47	371

Figure 7.9 classification report for SVC before tuning

	precision	recall	f1-score	support
0	0.45	0.46	0.46	106
1	0.56	0.64	0.60	182
2	0.23	0.14	0.18	83
accuracy			0.48	371
macro avg	0.41	0.42	0.41	371
weighted avg	0.45	0.48	0.46	371

Figure 7.10 classification report for SVC after tuning





















Club	MP	W	D	L	GF	GA	GD	Pts
1  Liverpool	38	32	3	3	85	33	52	99
2  Man City	38	26	3	9	102	35	67	81
3  Man United	38	18	12	8	66	36	30	66
4  Chelsea	38	20	6	12	69	54	15	66
5  Leicester City	38	18	8	12	67	41	26	62
6  Tottenham	38	16	11	11	61	47	14	59
7  Wolves	38	15	14	9	51	40	11	59
8  Arsenal	38	14	14	10	56	48	8	56
9  Sheffield United	38	14	12	12	39	39	0	54
10  Burnley	38	15	9	14	43	50	-7	54
11  Southampton	38	15	7	16	51	60	-9	52
12  Everton	38	13	10	15	44	56	-12	49
13  Newcastle	38	11	11	16	38	58	-20	44
14  Crystal Palace	38	11	10	17	31	50	-19	43
15  Brighton	38	9	14	15	39	54	-15	41
16  West Ham	38	10	9	19	49	62	-13	39
17  Aston Villa	38	9	8	21	41	67	-26	35
18  Bournemouth	38	9	7	22	40	65	-25	34
19  Watford	38	8	10	20	36	64	-28	34
20  Norwich City	38	5	6	27	26	75	-49	21

Figure 7.11 Premier League 2019/20 final table[15]

	precision	recall	f1-score	support
0	0.32	0.78	0.46	116
1	0.54	0.24	0.33	172
2	0.33	0.10	0.15	92
accuracy			0.37	380
macro avg	0.40	0.37	0.31	380
weighted avg	0.42	0.37	0.33	380

Figure 7.12 Classification report of predicting the results using Logistic regression over the 2018/2019 season.

	precision	recall	f1-score	support
0	0.38	0.27	0.32	91
1	0.34	0.55	0.42	119
2	0.28	0.28	0.28	82
3	0.06	0.02	0.03	53
4	0.00	0.00	0.00	13
5	0.00	0.00	0.00	10
6	0.00	0.00	0.00	3
accuracy			0.31	371
macro avg	0.15	0.16	0.15	371
weighted avg	0.27	0.31	0.28	371

Figure 7.13 classification report for FTHG using Logistic Regression

# MSc Project - Reflective Essay

<b>Project Title:</b>	Predicting Premier League results using Machine Learning
<b>Student Name:</b>	Tariq Kamal Eldow Ahmed
<b>Student Number:</b>	161134893
<b>Supervisor Name:</b>	Thomas Roelleke
<b>Programme of Study:</b>	Big Data Science (Msc)

## Strength

- We have looked at different machine learning techniques to compare their scores which can be reproduced and reused for a similar project.
- Using the moving average allows the model to improve over each game. By using the current match statistics and averaging it with older match statistics, there is no need to retrain the whole model again when predicting home goals scored.
- Hyperparameter tuning enabled us to understand the relative importance of tuning and how important it can be in optimising model's performance.
- Using feature importance, it enabled us to explore which features were seen as important by the logistic regression model. Also, it showed if EA's rating had any effect on predicting the results.

## Weakness

- There was an issue, relating to a club that was not on the historical database but was promoted in the season that we tried to predict.
- There was a limitation in score accuracy when predicting the results. This is likely due to database imbalance, meaning few games had score lines higher than 2 goals. Therefore, it was harder to predict such score lines.
- The data was trained from one specific website football-data.co.uk, meaning the results obtained were not verified by another dataset.

## Further work

- Improved data such as using both Premier League and championship results to include more Clubs such as Sheffield united, Brentford and Aston Villa.
- Using KNN to estimate the strength of Sheffield United by aggregating teams of similar strengths
- Trying different combinations of EA rating such as difference between attacking strength and goal keepers diving ability
- Using a weighted moving average instead of just a moving average could improve results.
- Using more complex machine learning techniques such as TensorFlow and neural network

## Relationship between practical and theoretical

The project looks at a classical challenge of applying machine learning to predict the result of a football match. It contains the uncertainty, incompleteness of data which have been challenges in predicting game results. Within the time provided I have tried to compare different machine learning techniques to predict the match results. For example, logistic regression is an algorithm, by default, used for binary classification however in my project I needed to have 3 classes for the result(lose, draw, win). This meant either using OVR which uses a binary classifier per class or multinomial logistical regression which directly predicts the probability the input belongs to one of the multiple class labels. OVR was found to be better performing and so was used. The One vs rest method was used as the dataset was

not too large, and so the results were compiled within a few seconds. Also, it was found that Decision Trees slightly outperformed Random Forest. Which contradicts the theory that Random Forest, is a more robust model than Decision Trees. This was likely caused by noise in the data. The tuned random forest model had 1200 trees which may not have been enough. Additional trees could have been used to reduce the error rate.

With more time, the bayes model results could have been further improved. I chose to use GridSearch across all models. Using bayes optimization, could have increased its score. However, with the values achieved in the test/training split, it is unlikely to be a large increase in performance. I believe so as I haven't come across a project with an accuracy higher than 56%. Therefore, I believe it's unlikely that using bayes optimization would lead to much higher results.