AI for Football (or Soccer) Analysis

By: Cristiano Gaudino

Supervisor: Dr. Alejandro Arbelaez
Second Reader: Dr. Derek Bridge

University College Cork
April 2021

Abstract

This project attempts to accurately predict the result of a given football (or soccer) match by focusing on achieving the following goals:

- Identifying relevant attributes that will influence the outcome of a given match.
- Studying the correlation between expert ratings vs. (Machine Learning) predictions.

The result in this case consists of three possible classes: Home Win, Draw, Home Loss. To accurately predict these results, a custom dataset was built using web crawling techniques. This allowed for a wide variety of publicly available team, and player, statistics to be used in training various models. The data used in this project, provided www.fbref.com, allowed us to marginally improve upon the accuracy of expert Bookmakers.

Various models were trained and tested until the best model was chosen, which ultimately had an accuracy of 60%. This model was then extensively tweaked and tuned, at which point we could begin comparing with the bookmakers. Using the odds given by multiple bookmakers, over multiple different seasons, it can be proven that the best bookmakers are 58% accurate.

Declaration of Originality

In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: *Cristiano Gaudino*
Date: 27/04/2021

Table of Contents

1 Introduction

Sports betting is a huge financial sector and when it comes to soccer, the industry is worth almost two billion Euro each year[1]. With bookmakers determining the odds for soccer matches, and bettors trying to gain a betting profit, there is a huge incentive for machine learning when it comes to predicting the outcome of these fixtures, and seasons.

Bookmakers spend a large amount of time setting the odds for each individual fixture, with a number of experts and machine learning techniques being used to do so. As such the information surrounding the reasoning for these odds is often kept secret, to protect the bookmakers that set them. We will attempt to uncover some of the key attributes that allow bookmakers to create these accurate predictions, as well as study the correlation between their expert ratings and machine learning techniques.

Many studies have been centered around a similar topic, however there is much uncertainty when it comes to predicting the outcome of these matches. For example draws are often thrown aside due the incredible difficulty that comes from predicting them. This often means that a large number of papers, such as Baboota and Kaur (2018) or Ulmer et al. (2013), are left with an accuracy below 60%. When comparing this to the bookmakers, we will try to consider whether they face these same issues, and how they are compensated for when setting the odds of each individual fixture.

2 Background & Related Work

Hessels (2018) attempts to build upon the work of Ulmer et al. (2013), by focusing on engineering a number of features. Using this set of new features, the most noteworthy being weighted form, the accuracies of 50% and 51% achieved using an SVM and a Random Forest respectively, were outperformed by Hessels (2018). In both cases, the accuracies were increased to 60% due to the newly engineered features, as well as an extended dataset that spanned over a number of years. It is interesting to note here that initially a smaller dataset had been used, however this was limiting the results to 55%, once the dataset was extended to include a much larger number of years it's full potential was achieved. In comparison to this, Hessels (2018) also attempted to extend the dataset by including information from multiple leagues, rather than just focusing on the Premier League. This however had the undesirable effect of reducing the accuracy, many leagues are played in different styles, thus the influential features of one league may not translate when applied to others.

Razali et al. (2017) managed to achieve a final accuracy of 75%, this accuracy is remarkably high when compared to what has been achieved by others. This is likely due to the usage of k-fold Cross-Validation, where 90% of the dataset was used as training data. This results in fixtures being present in both training and validation, which in this

[1] https://www.mirror.co.uk/sport/football/news/football-betting-worth-14billion-gambling-10498762.

case is an example of information leakage. Due to the nature of soccer matches, and the uniqueness of each individual match, it is important to prevent leakage as much as possible. If it is not prevented, then the final result will be overly optimistic, and will likely be less accurate than it could have been. When taking this into consideration, it will be important to use Holdout when splitting the dataset, this will allow for no crossover between training and testing.

Hessels (2018) references the shortcomings of the work they had completed. Most notably the lack of sufficient data for training and testing, as well as the lacking quality of said data. It also mentioned that choosing to compromise and select an existing dataset, rather than creating their own was one of the choices that was regretted by the end of the project.

3 Specification

The first set of specifications revolve around the dataset, a strong dataset is the foundation for any artificial intelligence system. First, the dataset will have to contain enough information regarding the current season that is being played. This pertains to potential sports betting opportunities, for which the model produced will need as much information as possible. This encompasses a large number of statistics, but can potentially include Number of Wins, Total Goals Scored, Average Possession Percentage per Game. These statistics will not only have to be available in abundance for the current season, but also multiple previous seasons. One season contains a total of 380 matches played between all teams, this alone would not be enough data to train and test the produced model. As such, the aim is to have four seasons for training and validation, and one season for testing.

Player statistics are as important as team statistics, as such this information must also be made available. For a given fixture, the lineups of both the Home and Away team must be present. If a team elected to not play their best players, it would affect the result of the game. As such the model must be made aware of any changes made to the lineups. Similar to team statistics, the statistics for each individual player in the league, must also be made available. This will give the model a better understanding of how well a given player may perform. Finally, similar to team statistics, this information must be available for a number of previous seasons.

Player statistics alone, could potentially account for upwards of 100 features. As such, the final model used for predictions must be able to handle a large number of features. The model must also be able to make predictions at a relatively high speed, if a user has to wait a long period of time to get a prediction, they'll simply move on to another source.

4 Design

4.1 Dataset

A large number of soccer related datasets are available online. These datasets cover a wide range of leagues, and can date back a number of years. They also cover a variety of information, some storing betting odds for every fixture, while others focus on storing as much player related data as possible. The problem however, is that there is no publicly available dataset that covered all of the specifications previously discussed. Rather than reducing the number of specifications, potentially leading to no player data being included, the choice was made to create a dataset for this paper. This allows for all specifications to be fulfilled, as well as allowing for total control over which features are present in the dataset.

All of the data used to create the dataset was gathered from fbref.com, there is an extensive amount of information available on the website for both team and player statistics. This information is also available for a number of seasons.

4.1.1 Squad Statistics

For each season, the data is presented in a number of categorised tables, which are available as CSV files. Nine of these tables will be used for creating the dataset, each of which contain unique information regarding the season. These tables contain squad statistics under the categories of defense, shooting, passing, possession, and goalkeeping. A table of all 380 fixtures within the season is also included, this contains the date of a given match as well as the Home and Away team that will be playing.

Figure 4.1 displays the information available in a single row, from the Squad Shooting table. A number of the features available are largely redundant, or are linearly correlated with each other, an example of this being Shots per 90 Minutes. However, some features such as goals scored, or shots on target, may be helpful in predicting the result of a fixture.

Heading	Description	Value
Squad	The name of the team	Arsenal
# Pl	Number of players used in games	28
Gls	Goals scored	69
Sh	Total shots taken	464
SoT	Shots on target	158
SoT%	Percentage of shots that are on target	34.1
Sh/90	Shots taken per 90 minutes	12.21

SoT/90	Shots on target per 90 minutes	4.16
G/Sh	Goals per shot taken	0.14
G/SoT	Goals per shot on target	0.41
Fk	Shots from free kicks	11
Pk	Goals from penalty kicks	4
xG	Expected goals (calculated by fbref.com)	57.2

Figure 4.1. A sample row from the Squad Shooting table

As stated, each of these tables provides unique information related to each of the teams. From each of these tables, a number of features were selected to be included in the created dataset. Figure 4.2 displays the chosen features, from each of the available tables. Each of the features displayed is a culmination of a team's statistics over the current season, meaning that Goals Scored is all of the goals a team has scored over the season. A number of features were not included, most notably the "per 90 minute" features.

Table Name	**Selected Features**
Fixtures	Date; Home Team; Away Team; Result (Being either W, D or L)
League Standings	Number of Wins; Number of Draws; Number of Losses; Points
Squad Statistics	Possession; Goals; Assists; Red Cards
Goalkeeping	Goals Allowed; Shots on Target Allowed; Saves; Clean Sheets
Shooting	Shots; Shots on Target; Goals per Shots; Goals per Shot on Targets
Passing	Completed Passes Percentage; Key Passes; Passes into Oppositions Third; Progressive Passes
Goal and Shot Creation	Shots Created; Passes Leading to Shots; Dribbles Leading to Shots; 2nd Chance Shots; (The same stats for goals are also included)
Defensive Actions	Tackles Won; Blocks; Interceptions; Errors Leading to an Opponent's Shot
Possession	Live-Ball Touches; Miscontrols

Figure 4.2. All of the selected features, from the tables provided by fbref.com

4.1.2 Player Statistics

As per the outlined specifications, for every fixture, the lineups for both the home and away team must be present. The statistics for every player in the league are made available in a table, similar to the squad statistics, by fbref.com. However, the lineups are not stored alongside the fixtures, and as such must be acquired through different means. A URL for a match report page is provided for every fixture, this page provides a number of statistics, but most notably it provides the lineups for both teams. Using this URL and the requests library in python, the raw HTML from each match report webpage can be acquired. The BeautifulSoup library allows for this HTML to be converted to a string, at which point the lineups can be easily parsed out. The player statistics, for each team, are then gathered using the resulting list of players.

A large amount of data is provided by the player statistics table. However, the decision was made to remove most of this information to reduce the size of the final dataset. Figure 4.3 displays the features used for each player, the exception being the Goalkeeper, for which only minutes and matches played are used. The dataset is also anonymous, due to the exclusion of player names in the final dataset. This decision was made to further reduce the size of the dataset, there are almost 500 players in the Premier League and one-hot encoding each player would not be feasible.

Heading	Description	Value
Matches Played	Total number of matches played by the player	36
Minutes Played	Total number of minutes played by the player	3,043
Goals Scored	Goals scored by the player	12
Assists	Assists by the player	4

Figure 4.3. A sample row from the Player Statistics table

4.2 Predicting

The chosen dataset will have to be modified to an extent. This includes feature preprocessing, and engineering new features to give the model a better understanding of the data present. A number of models were trained on the dataset for a variety of reasons, for example AdaBoost was used as it is a strong ensemble when dealing with binary problems, whereas Decision Trees are strong when dealing with a large number of features.

kNN is an instance based classifier that uses euclidean distance to calculate the most similar examples in the dataset, and then uses these examples to make a prediction. While it is not complex, it can still be quite accurate in most cases, as such this model was trained and tested. The two main hyper-parameters to discuss here are the number of

neighbours and whether it is weighted or not. The neighbours are used to make the final prediction, based on majority vote, too few neighbours may leave the prediction overly influenced by its neighbours, whereas too many neighbours will result in the prediction just being the majority class of all the examples in the dataset. Weighted kNN results in the nearest neighbours having more influence than further neighbours.

Decision Tree is a model based classifier, that uses nodes with criteria based on a number of learned factors to split possible predictions. These nodes are defined by a number of hyperparameters, most notably the criterion, which measures the quality of a split, and the splitter. The splitter states which strategy to use when splitting at each node, either being the best strategy or a random for the best random split. The criterion has two possible values, gini and entropy, the Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The highest Gini score is 0.50 while the highest entropy score possible is 1.0. (Wilson, 2018). Finally a max depth defines the maximum number of splits from the uppermost node.

Random Forest is an ensemble of Decision Trees which uses random subsets of features and majority voting to make predictions. The number of trees present is defined by a hyper-parameter known as n-estimators. As with decision trees, the hyper-parameters of criterion and max depth are present. The final hyper-parameters to discuss relate to bootstrapping, this is a resampling technique that can be set to true or false, if set to false then the entire dataset is used for every tree. If set to true, then bootstrap samples will be used when building the trees. The random state parameter, controls the randomness of these bootstrap samples, as well as the features to consider when looking for the best split at each node.

Eventually it became clear that draws were almost impossible to predict, with many models opting to never predict them at all. As such, an AdaBoost classifier was trained and tested as this model is well known for it's high performance on binary classification problems. By default, the model is initialised with a Decision Tree of max depth 1, the n estimators hyper-parameter then defines the number for which boosting is attempted. There is also a learning rate that is applied to each classifier at each iteration, a trade-off exists between both of these hyper-parameters, a lower learning rate will require a much higher number of estimators to all for continued improvement of the model. Once it became clear that logistic regression was quite strong, another AdaBoost classifier was trained, this time with logistic regression as the base estimator. The result of this was good enough, however it was not better than the default AdaBoost classifier, or logistic regression by itself.

Logistic regression is a model based classifier, that makes use of the one-vs-rest scheme to support multi-class problems. The model utilises the l1 or l2 norms to implement regularisation. The l1-norm, or lasso regression, uses the absolute sum of values for beta, whereas the l2-norm, or ridge regression, uses the square of the values. In the case of the l1-norm, this can result in features being ignored as the corresponding beta is reduced to 0. While the l2-norm does not reduce betas to 0, they get very close, effectively achieving the same result. Thus, in some sense feature selection is also being performed here. The

regularisation strength is controlled by a hyper-parameter C, with smaller values specifying stronger regularisation. In the case of multi-class, the softmax function is used at output. In this case this will give 3 numbers, the probability that the prediction belongs to a certain class, these 3 numbers are then reduced such that they sum to 1. Finally it is possible to use one-vs-one instead of one-vs-rest, this comes with a higher training cost but can potentially lead to a better result.

5 Implementation

5.1 Parsing the Dataset

To build the dataset for this paper, the Parser first reads in the fixtures CSV file. For each fixture, the relevant squad and player stats will have to be gathered and added to the outputted example. As such, the Parser begins by looping over the fixtures. The home and away team names are passed to a separate function, gatherStats, this function serves as the gathering point for squad and player statistics. The process is identical for both teams, and focuses first on the squad statistics.

5.1.1 Parsing the Squad Statistics

The squad statistics are parsed using a set of methods, the first being the getStats method, which takes in a team name. This method contains the paths to all relevant CSV files, as well a list of indices relevant to each file. This list of indices corresponds to the desired features for each file, all of which were previously discussed. The reason for doing this is to make the code much more readable, with the csv files and desired columns all defined in this function, it is easy to make adjustments and modify the information contained in the final dataset. The team name, CSV path and list of indices are all passed to another function, getSpecificStats. This function will open the given file, and return a list of statistics that correspond to the given team name and columns. This information is then passed back up the line, to be combined with the player statistics.

The headings are treated in the same manner as the statistics, the main difference being that some headings must be modified such that they remain unique. The headings are the first line that get parsed, as such they follow the same path as standard statistics. Once the getSpecificStats function is called, any names that conflict can be modified slightly. The most notable change here is the duplication or all headings, and the appending of "_Home" or "_Away" to the end of each heading.

An oversight was later discovered regarding this approach. For each fixture, the functions will loop over all 9 CSV files twice, once for the home team and once for the away team. This can be a slow process, and a much more efficient approach would have been to convert all of the CSV files to hash tables at the start of the parser. Hash tables have an average lookup time of O(1), or constant time, this would greatly reduce the overall runtime of the parser. This approach was later implemented when parser player statistics.

5.1.2 Acquiring the Lineups

As previously mentioned, the lineups for each fixture are not stored within the CSV files. However, a link to the match report is available on the webpage where the fixtures CSV file is displayed. This link can be obtained through a number of approaches, such as using a web crawler, however the simplest option is to get the raw HTML data from the web page and parse it.

The “requests” library is used to get the raw HTML text from the fixtures web page. The “BeautifulSoup” library allows for the HTML text to be converted to a string. The “find_all” method, provided by BeautifulSoup, takes in a HTML tag and returns the text within it. In this case, it is used to get the first occurrence of a table from the text.

In HTML, rows are defined by the “tr” tag, and columns by the “td” tag. With these two tags, the “find_all” function can be invoked on the table to build a list of rows. This functionality is encapsulated in it’s own function, parseHTMLTable, and is reused when acquiring the team lineups. This process of building the fixtures from the HTML is initially run once and stored. This is then passed to any functions that require said information.

Once the correct fixture is found, the Home and Away team name can be used within a simple regular expression. If both team names are found within the HTML tags and other unwanted characters, then the fixture is correct and the match report URL can be retrieved.

The above functionality to acquire the fixture URL is reused to retrieve the lineups. Once the match report URL has been retrieved, the “requests” & “BeautifulSoup” libraries can be used to get the HTML content and convert it to a string. The previously mentioned parseHTMLTable function can then be reused to acquire the 2 tables containing the lineups for each team. The list of player names can then be retrieved using a regular expression.

5.1.3 Parsing the Player Statistics

As with the squad statistics, the headings for the player features must be unique. A simple solution to this is to create an array, populated with the initial starting headings. For each outfield player there are 4 headings, all of which were previously mentioned. This array can be modified, such that a number can be added to each of the 4 headings for the 10 outfield players. The goalkeepers only have 2 features, and once these are accounted for, there are 60 features total. The array is then duplicated such that “Home” and “Away” can be appended. These 2 arrays can finally be concatenated and returned.

If done incorrectly, the computation time for parsing the player statistics can become far too large. To combat this, the player statistics CSV is initially read in as a hash table, as previously mentioned this accompanies an average constant lookup time. To further combat these issues, unlike squad statistics being calculated individually for each team, player statistics are gathered for both teams. This prevents the same match URL page being visited twice, and should further help to reduce any computation times.

The statistics for the lineups are gather using a function that takes in the following:

- Home and Away Team names.
- The hash table of player statistics.
- The rows of fixtures, wherein each fixture is a string of HTML text, as previously discussed.
- The list of indices, representing the desired columns.Building the Model.

Eight empty arrays are initialised, four for each team to represent the four main positions of goalkeeper, defender, midfielder, and forward. The lineups are retrieved using the functions previously discussed, such that for each player in the lineup, the hash table will be queried. The appropriate statistics are then retrieved using the indices passed in. Finally, after all player's statistics have been gathered, the arrays can be concatenated and returned.

The result of this is an anonymous dataset, that contains both squad statistics and player statistics based on the starting lineups for both teams.

5.2 Feature Preprocessing

For the created dataset, the target feature is the Full Time Result (FTR), which has the possible values of W, D, or L (Home Win, Draw, or Home Loss). Using the "pandas" library in python, the correlation of each feature in the dataset with the FTR, can be calculated. Overall the correlations remained quite low, with the higher numbers staying around 25%. However, a large number of features were redundant, such as Red Cards which had a correlation of 3%. As such, these features were removed in favour of reducing the complexity of the dataset.

A number of basic methods to engineer new features were attempted, such as getting the square root of a feature, however these all resulted in quite low correlations. Figure 5.1 displays the engineered features that were implemented into the dataset, all of which use the average of a stat over the number of wins a team has.

Engineered Feature	**Correlation**
Clean Sheets per Win by the Home Team	-16%
Goals per Win by the Home Team	-18%
Goals per Shot on Target, Win by the Home Team	-20%
Goals per Shot on Target, Win by the Away Team	14%
Shots on Target per Win by the Home Team	-18%

Figure 5.1. A table displaying correlation of engineered features.

An engineered feature that was prevalent in a number of related papers, such as Baboota and Kaur (2018), was recent form. This would represent how many wins a team has over the last number of games. This would've been a highly correlated feature, however implementing it would've meant going back and redoing the dataset creation. Thus, requiring that all of the datasets be parsed again for each season, which is a time consuming process. Due to time constraints this feature was not included.

5.3 Training & Selecting Models

With the result of a football match being either a Home Win, Draw, or Away Win, a suitable multi-class classifier model is needed to handle this problem. A number of related papers have achieved varying degrees of success, with various models for this problem. Baboota and Kaur (2018) achieved an accuracy of 57% using a random forest model, whereas Hessels (2018) achieved an accuracy of 60% using a support vector machine. Finding the appropriate model for this problem meant that a number of them had to be trained and tested.

A number of models require that the data present be scaled & normalised. As such each model was trained using 3 separate scalers, the Standard Scaler, the Robust Scaler, and the MinMax Scaler. While the standard scaler follows standard normal distribution, the MinMax scaler will scale all features between 0 and 1. Whereas the Robust Scaler, named due to its robustness to outliers, scales data between the 1st and 3rd quartile, also referred to as the interquartile range. Each of these has their own benefits and work well with different models, all three were included when optimising the hyper-parameters of each model.

Hyper-parameter optimisation is a long process and can be done through a number of methods. For this paper, both grid search and random search were used. Grid search, while much more extensive, is also very time consuming and as such for models like an SVC, random search was initially used. Once random search narrowed down the result, grid search was then used with similar hyper-parameters to get the best values possible, with the time available. Eventually, once one or two began to show as the favourites, these models could then be extensively grid searched over a number of days.

When training and validating the accuracy of the models, it is important to decide whether to use k-fold cross validation, or to use holdout. Holdout is used for large datasets, it guarantees that no data will be reused between training and validation. Whereas k-fold favours smaller datasets, and by reusing examples for both training and validation, a smaller dataset can overcome the issues that accompany a lack of data. For this paper, holdout was used as there is enough training data available to avoid any crossover, however due to the unique properties of each fixture, it was also important to avoid having examples present in both training and validation. Holdout can also be used in a stratified fashion, which allows for the distribution of classes to remain the same when splitting up the dataset.

The best iteration of the kNN model, used 50 neighbours and was not weighted, weighted kNN in this case led to a huge amount of overfitting. The resulting training and validation

accuracy of kNN, displayed in Figure 5.2, is 57% accurate for both training and validation.

When training the Decision Tree model, the best choice for each of the hyper-parameters was a max depth of 3, entropy as the criterion and the best split over random. These hyper-parameters, as shown in Figure 5.2, result in a training and validation accuracy of 58%, a slight improvement over kNN.

The best version of the random forest model trained, used 6 subtrees, all of which had a max depth of 5, and used the entropy criterion when measuring the quality of a split. As shown in figure 5.2, this resulted in a training accuracy of 62% and a validation accuracy of 58.5%. This model is overfitting slightly, however, it was one of the two models that ended up being extensively grid searched over a number of days. The result of this is discussed in the Results section.

The best iteration of the AdaBoost classifier had a learning rate of 0.181, 71 estimators and a random state of 2, random state in this case operating in the same fashion as with Decision Trees. The results of this, as shown in Figure 5.2, are a training accuracy of 61%, and a validation accuracy of 57.5%, which is overfitting by some margin. This classifier takes a long time to train, as such the issue with overfitting could not be reduced any further.

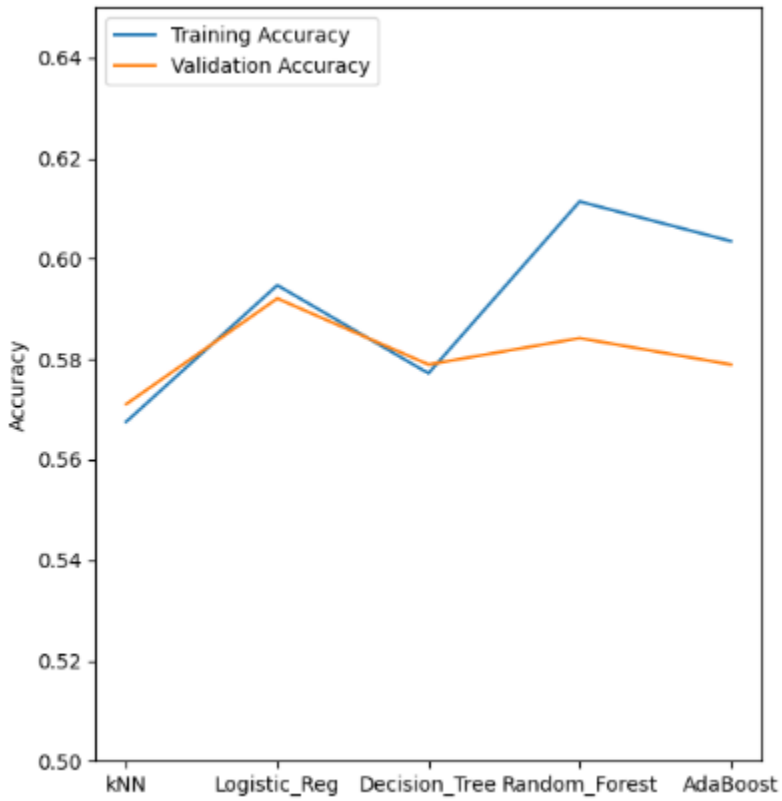

Figure 5.2.
A graph displaying the training and validation accuracies of each of the best models.

In the end, logistic regression ended up being the most promising of all the models. Utilising the one-vs-rest scheme and the l2-norm, with a value of 0.002 set for the hyper-parameter C, a training accuracy of 59.4% and a validation accuracy of 59.2% were achieved, both of which are shown in Figure 5.2.

6 Results

Based on the Model Selection results, it was decided to proceed with Logistic Regression and Random Forest, both of these models were extensively grid searched over a number of days to get the best possible hyperparameters. Figure 6.1 illustrates the resulting confusion matrices for these models, it's clear that both still remain very close in terms of accuracy at 60%, however the Random Forest model was still overfitting slightly, whereas Logistic Regression had no issues with under/overfitting. Based on this Logistic Regression was chosen as the best model.

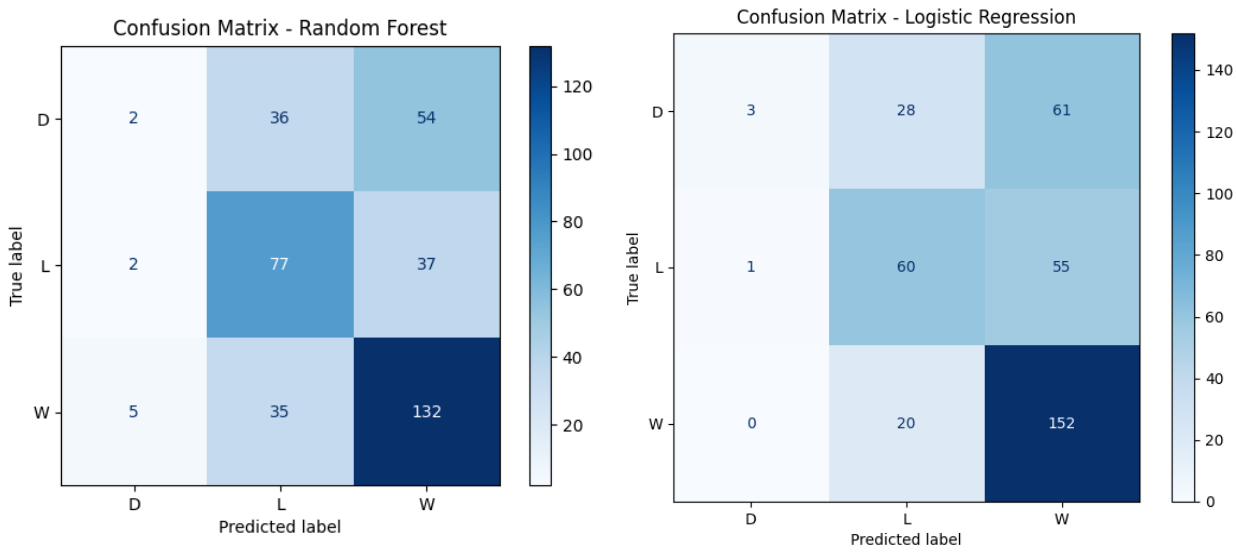

Figure 6.1.
Two Confusion matrices, representing Random Forest & Logistic Regression

Bookmakers do not not make their prediction models, or the accuracy of them publicly available. As such to use the bookmakers accuracy as a baseline this had to be calculated from a dataset of odds provided by bookmakers for a given fixture. The odds given by Bet365, one of the largest British gambling companies, were used to calculate this baseline. The result of this when using the odds from the previous 4 seasons of Premier League football is an accuracy of 55%, whereas when using just the previous season the accuracy is 58%. In-line with the many models that were trained for this problem, the bookmaker never predicts draws and instead elects to never classify any predictions as draws.

6.1 Experiment 1

To compare the results of using the created dataset with the bookmakers, a dataset was created using the bookmakers odds. This dataset includes odds given by several top bookmakers such as Bet365, Ladbrokes and more, with three seasons being used for training/validation and one for testing. As the best model trained off the created dataset was Logistic Regression, a Logistic Regression Classifier was also trained off the odds dataset. Depicted in Figure 6.2 is the resulting confusion matrix of the Logistic Regression model trained with the bookmaker odds dataset. With an accuracy of 58.5% and some slight overfitting it is worse than the best model, however not by much. It is also evident that draws are still disregarded by the bookmakers as illustrated by the confusion matrix.

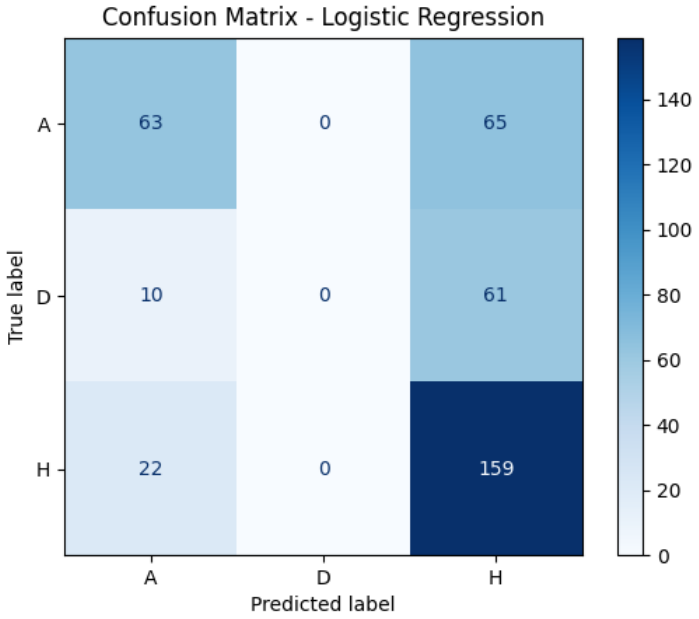

Figure 6.2.
A confusion matrix for a model built using bookmaker odds as training data

6.2 Experiment 2

Using a basic model, the improvements made by feature preprocessing and engineering can be observed. It should be noted that the test set, in this case the 19/20 Premier League season, is not included in this test to prevent any sort of bias. In this case, a logistic regression model was trained on three separate datasets, one representing the initial unaltered dataset, the second representing the dataset after removing the redundant features, and finally the dataset after including the engineered features. Figure 6.3 displays the training and validation accuracies of all three instances. The unaltered dataset is underfitting slightly, due to the large number of features, while the removal of the redundant features seems to have drastically increased the amount of overfitting by the model. The final iteration of the dataset, while only marginal, has improved over the initial iteration, with a slight reduction in underfitting as well.

Dataset	Training Accuracy	Validation Accuracy
Unaltered	56.5%	58%
Post Feature Selection	58%	54%
Final Iteration (Includes Engineered Features)	58%	58.5%

Figure 6.3. A table displaying the results of feature preprocessing and engineering

7 Conclusions & Future Work

It has been shown in this paper that when it comes to predicting the outcome of fixtures, bookmakers are faced with a large number of problems that must be accounted for. With bookmarkers being 50% accurate when predicting away victories, and almost 0% accurate when it comes to draws, it's clear that there is a huge amount of uncertainty on the topic. Randomness is hard to account for when predicting these results, and as such bookmakers opt to always favour a definitive win or loss when setting their odds. These odds will always be set to favour the bookmakers, and while the final iteration of the model produced for this paper was 60% accurate across all results, the bookmakers will have accounted for this with the most likely scenarios having much more favourable odds. Even as our model achieves an 85% accuracy when predicting home victories, this is still only a marginal improvement over the bookmaker's 70% accuracy.

However, despite these shortcomings, it is clear that machine learning can still compete within this industry. A logistic regression model trained over a number of days, resulting in an accuracy that marginally outperforms the bookmakers, shows how promising the field of machine learning is when it comes to sports betting. Despite the issues and challenges that come with soccer predictions, for example draws being impossible to predict, the model produced by this paper still managed to achieve a result that is on par with, if not marginally better than, an industry worth almost 2 billion Euro each year.

7.1 Future Work

At its current stage, the model will have to be manually fed the required inputs for making predictions. A future step would be to modify the parser to take in the date of a fixture and the 2 teams playing. This would then allow the parser to run as normal, as it would when generating the information for each fixture, the main difference being that it would simply stop after one iteration. The result of this would be a system that allows users to select any match and get back a prediction made by our best model.

One of the shortcomings of the current iterations comes from the start of new seasons. At the start of a new season, there will be a lack of information available regarding the teams playing. At the moment the dataset is built using only information from the current season, this is strong when predicting games towards the latter half, however this means

that predictions will be overly influenced by the results of just a few matches at the start of the season. Implementing statistics from previous seasons, would allow for the final prediction to not be solely influenced by the current season, thus solving this issue..

References

FBRef. (2021). FBRef.com. Football/Soccer Statistics for both teams and players. Retrieved from https://fbref.com/en/comps/9/Premier-League-Stats

Football-Data. (2021). Football-Data.co.uk. European Football Results and Betting Odds [database file]. Retrieved from http://www.football-data.co.uk/englandm.php

Baboota, R., & Kaur, H. (2018). Predictive analysis and modelling football results using machine learning approach for Premier League. *International Journal of Forecasting*

Hessels, J. (2018). Improving the prediction of soccer match results by means of Machine Learning. *Tilburg University, Netherlands*

Wilson, J. (2018). Gini Impurity and Entropy. medium. Retrieved 04 20, 2021, from https://medium.com/@jason9389/gini-impurity-and-entropy-16116e754b27

Ulmer, B., Fernandez, M., & Peterson, M. (2013). Predicting Soccer Match Results in the Premier League. *Doctoral dissertation, Stanford.*

Razali, N., Mustapha, A., Yatim, F., & Aziz, R. (2017). Predicting Football Matches Results using Bayesian Networks for English Premier League (EPL). *IOP Conference Series: Materials Science and Engineering (Vol. 226).*