EVALUATION OF THE CURRENT STATE OF FOOTBALL MATCH OUTCOME PREDICTION MODELS

Thiebe SLEEUWAERT
Student ID: 01302061

Promotor: Prof. Dr. Christophe Ley
Tutor(s): Prof. Dr. Christophe Ley

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of Master of Science in Statistical Data Analysis.

Academic year: 2019 - 2020

GHENT UNIVERSITY

Gent, September 4, 2020

The promotor, The author,

Prof. Dr. Christophe Ley Thiebe SLEEUWAERT

ACKNOWLEDGEMENTS

First and foremost, I would like to thank the promotor of this master dissertation, Prof. Dr. Christophe Ley, for allowing me to work on this exciting subject. In these unusual times of the coronavirus pandemic, it was not always easy to discuss some approaches and results with your fellow students or professors. Still, Prof. Dr. Christophe Ley adapted to this situation and granted me the guidance I needed.

Secondly, I would like to thank both Prof. Dr. Lars Magnus Hvattum and Dr. Hans Van Eetvelde. Prof. Dr. Lars Magnus Hvattum provided the necessary data to include the plus-minus ratings in this thesis, and Dr. Hans Van Eetvelde shaped this data in the right format to work with.

Last but not least, I would like to thank everyone that made my unusual educational path throughout the university of Ghent possible. Starting out in 2013 as a bachelor student in biology and finishing in 2020 as a MSc in Statistical Data Analysis.

CONTENTS

ABSTRACT

Sports betting knows a long history and has always excited the fans. In recent times, football is one of the most popular sports worldwide. Thus the forecasting of football matches is prevalent. Aside from betting, forecasting the outcome of football matches is also relevant for sports journalists and decision-makers within the sport. Many statistical models have been proposed to predict football match outcomes. These models usually incorporate or estimate the strengths of the opposing teams relative to each other. Two main types of models that predict football matches are recognised, namely the result-based and goal-based approaches.

The result-based models predict the outcome classes (homewin/draw/awaywin) of the football matches directly. A simple example of these result-based models is the ordered logistic regression. At the same time, more advanced methods include machine learning techniques, such as random forest classification.

The goal-based alternatives first estimate or assume a suitable distribution for the goals scored by the opposing teams. From those distributions, the outcome classes are then derived. Simple examples include the independent Poisson model, and more advanced examples use machine learning techniques, such as the random forest regression model.

Many models are conducted on different scales. The first scale is how various scholar use different scoring rules to evaluate the models, which makes it challenging to interpret the results over multiple articles. A second scale is that various scholars evaluate their models on diverse leagues. Diverse leagues have distinct competitive levels, which influence the randomness and predictability of the matches. The last scale is how various scholars use vastly different training and testing protocols.

We aim to compare the performance of the current literature, by using a mixture of both simple and more complex models, from both the result-based and goal-based approaches. To circumvent the problems of different scales, we first bring all the models on a uniform training and testing procedure. Secondly, three essential scoring rules (ranked probability score, Brier score and ignorance score) are used to evaluate and rank the performance of the included models. The data for all the models originates

from the English premier league, which circumvents the problem of using diverse leagues.

In total, we compared 12 models to each other: two naive methods, named the uniform and frequency model, three order logit regression approaches, from which two were based on the ELO rating system and one based on the plus-minus rating system, three goal-based models, namely the independent Poisson, the bivariate Poisson and Weibull count model and four machine learning models, namely two random forest approaches, a gradient boosting approach and a hybrid random forest approach. From the two random forest models, one used a result-based approach and the other a goal-based approach.

Our results show that more sophisticated machine learning models have better predictions compared to more simple alternatives. In particular, the gradient boosting model from Baboota and Kaur (2019) had the best performance overall the scoring rules. For this model, we report an ignorance score of 0.933, a Brier score of 0.353 and a ranked probability score of 0.132. We were not able to distinguish between the performances of the result-based and goal-based models. However, goal-based models hold the most promising results. Our results also illustrate the importance of informative and qualitative features, e.g. the models by Baboota and Kaur (2019) had a mixture of both historical features and features that captured the recent performance of the team.

Although our included models cover some key contributions and recent additions to this field of statistical modelling, there are still a lot more models remaining. We suggest that instead of creating new and highly sophisticated models and features, the scholars in this field should focus on ensembling established models and use comparative studies to rank the current literature.

CHAPTER 1

INTRODUCTION

In this chapter, the first section gives a brief introduction to the history of sports betting. The following section presents a summary of the current literature concerning football match prediction models. After that, a section dedicated to the most popular football leagues and scoring rules is given. Finally, the last section specifies the problem statement and objectives.

1.1 A brief history of sports betting

Sports betting and sports forecasting have been around for as long as sports events have existed. Records of sports betting go back as far as 2000 years ago. It originates from the ancient Greeks who used to bet on athletic competitions during the Olympics and from the ancient Roman empire where betting on gladiator fights occurred. In those empires, sports betting eventually became legal and consequently spread to neighbouring kingdoms. During medieval times, religious leaders abolished sports betting, which forced it to stay underground, where it continued to grow in popularity (Milton, 2017).

Later in the United Kingdom, sports betting became again increasingly popular and mainstream in the form of outcome betting on horse races. People could place their bets through bookmakers, or so-called "bookies". Presumably, the first bookmaker in the United Kingdom opened in the 1790s (Munting, 1996). Throughout the 20th century, multiple countries legalised sports betting again, e.g. 1931 in Nevada and 1961 in The United Kingdom. In 1994, the country Antigua and Barbuda passed laws that allowed enterprises to apply for online betting licenses and so the modern form of online sports betting was born (RightCasino, 2014).

In recent times, football (or soccer) has become one of the most popular sports in the world, so sports betting on football matches is also prevalent. Betting on football matches is estimated to take up 70% of the sports betting market (Keogh and Rose, 2013). The most popular form of betting is outcome betting, where the aim is to predict which of the two competing teams will win the match.

The prediction of which team will win the match excites football fans all around the world. Online bookmakers have created a business out of this excitement and made it into a vast and competitive industry. The prediction of football match outcomes also gives valuable insights for sports journalists and decision-makers within the sport. The following section gives a review of the current literature concerning the prediction of football match outcomes.

1.2 Literature review

There are numerous studies regarding the prediction of football match outcomes. Stefani (1977) predicted football matches by using a least-squares model that rated the strengths of both competing teams relative to each other. The trend of first estimating the relative strengths of the competing teams is still present in numerous recent studies. Estimating the relative strength of a team can be done by a multitude of methods and is usually conducted on a team-based level. Recently, with the increase in available data, researchers have rated players on an individual-level and derived the ratings of a team from those individual player ratings (Arntzen and Hvattum, 2020).

Most statistical models that aim to predict the results of football matches are categorised under two main types. The first type of models aims to estimate the distribution of the goals for both competing teams. From those distributions, the outcome classes (win/draw/loss) are then indirectly derived. We will refer to these types of models as *goal-based* models. The second type of statistical models aims to predict the outcome classes directly. We will refer to these types of models as *result-based* models. The following subsections give an overview of the methods that hold notoriety in this field.

1.2.1 Goal-based models

Goal-based models aim at estimating the goal distribution of the opposing teams. These models first assume a suitable count distribution for the football goals scored by each team. One of the most frequently used approaches assumes that the goals follow a Poisson distribution.

Independent Poisson

The independent Poisson model, first proposed by Maher (1982), assumes that the number of goals scored by each team follows a Poisson distribution and that these distributions are independent from each other (Katti and Rao, 1968). The following formula gives the Poisson density function:

$$P(X = x|\lambda) = \frac{\lambda^x}{x!}e^{-\lambda} \tag{1.1}$$

This formula shows the probability of observing x number of goals given λ, with λ equal to the expected number of goals for a given team. We will refer to this parameter λ as the rate intensity parameter.

Most statistical models first estimate λ by constructing it to represent the relative strengths of the competing teams. For example, Ley et al. (2019) used $\lambda_{i,m} = exp(c + (r_i + h) - r_j)$ to estimate this parameter, where r_i and r_j represent the relative strengths of respectively the home and away team, and h represents the effect of playing as the home team. Chapter 2 gives a more thorough explanation of this model.

Dependent Poisson

Experts in the field mostly agree that the assumption of independence between the goal distributions of both competing teams is flawed. These goal distributions have some apparent dependence between them, since the estimation of the λ parameters usually involves the relative strengths of both competing teams (Ley et al., 2019). Additionally, in team sports, it is reasonable to assume that the goals made by each team are dependent since both teams interact during a match (Karlis and Ntzoufras, 2003).

Dixon and Coles (1997) extended the basic independent Poisson model by including an indirect correlation term between the goal distributions of the competing teams. Dixon and Coles (1997) identified this correlation to be slightly negative. The correlation term is indirect since it ignores the direct correlation between the intensity parameters λ of the opposing teams.

Karlis and Ntzoufras (2003) used a bivariate Poisson model with a direct dependence term between the goal distributions of the competing teams. This bivariate Poisson model has the advantage that each goal distribution still follows a Poisson distribution marginally. A deeper explanation is given in section 2.3.3 - Bivariate Poisson.

There is a multitude of other methods available which introduce some form of dependence to the basic independent Poisson model. Some examples include McHale and Scarf (2011) who extended the independent Poisson with copula dependent structures. Boshnakov et al. (2017) presented a Weibull interval-arrival-time-based count process with a copula to model the number of goals for each team. We will discuss this model in more detail in section 2.3.4 - Weibull count model. The Weibull count model, and many other models, assume different distributions to model the goals made by the opposing teams. Some other examples include negative-binomial distributions, gamma-Poisson distributions and zero-inflated-Poisson distributions.

Skellam distribution

One of the advantages of using Poisson based models is that the Skellam distribution can be derived from it (Skellam, 1946). The Skellam distribution, or Poisson difference distribution, is the discrete probability distribution of the difference between two Poisson random variables (Ley et al., 2019). If we consider $GD_m = G_{i,m} - G_{j,m}$ as the difference between the expected goals scored by team i and team j during match m, then the probability of a win of team i over team j is calculated as $p(GD_m) > 0$. The probability of a draw and loss is calculated as respectively $p(GD_m) = 0$ and $p(GD_m) < 0$.

1.2.2 Result-based models

Although modelling the goal distributions is the most frequently used approach to predict football match outcomes, other types of statistical models with this aim exist. Namely, result-based models that directly predict the outcome classes. These models usually apply some form of ordered logit or probit regression. Recently, with the increase in available data and advanced computational algorithms, other types of result-based models have been proposed.

Directly modelling the outcomes does not indicate anything significant about the estimated goal difference between teams, which makes these result-based models milder in their assumptions and usually they have fewer parameters to estimate (Egidi and Torelli, 2020). A potential downside of these models is the overestimation or underestimation of the relative strengths of the competing teams since these models do not include the actual goal difference to derive these strengths (Egidi and Torelli, 2020).

Regression approaches

One of the earliest articles concerning result-based models, comes from Koning (2000), where a probit regression model was used to estimate the outcome classes directly. Goddard (2005) also used a probit regression model and compared it to the bivariate Poisson regression models. The article reports that the best performance was achieved by a hybrid model that combined a result-based dependent variable with goal-based lagged performance covariates. However, the differences among the models were small, and thus both approaches are considered relevant.

Hvattum and Arntzen (2010) used an ordered logit regression on the difference of the opposing teams ELO ratings and reported that the predictive performance of this model worked better than the result-based models of Goddard (2005). Hvattum (2017) reports that these logit regression models had difficulties predicting draws. To circumvent this problem, Egidi and Torelli (2020) used multinomial regression models with subtracted factors to inflate the probability of draws. They also compared goal-based and result-based approaches and found that the multinomial regression models were slightly lower in predictive performance when looking at the Brier score as a scoring rule, compared to the goal-based alternatives. However, the differences were again insignificant.

Recently, Arntzen and Hvattum (2020) also used an ordered logit regression model based on the plus-minus ratings of the players. The vast increase in data availability has made it possible to estimate such individual player rating. The authors report that the ordered logistic regression model based on the plus-minus ratings outperforms the ordered logistic regression model based on the team-based ELO ratings. However, when both covariates are combined, the predictive performance is significantly enhanced.

Thurnstone-Mosteller and Bradley-Terry models

Both Thurnstone-Mosteller (Mosteller, 2006; Thurstone, 1927) and Bradley-Terry (Bradley and Terry, 1952) type models where used successfully by Ley et al. (2019) to directly model and predict football outcome classes. These models predict the outcomes of pairwise comparisons by using latent variables.

Machine learning techniques

Recently, machine learning techniques have been used to predict the outcome classes of football matches directly. Joseph et al. (2006) showed that Bayesian nets out-

performed other supervised machine learning classification models such as decision trees, naive Bayes and k nearest neighbours. Constantinou et al. (2012) proposed a Bayesian network to predict the outcome classes of a match. In a follow-up study, Constantinou and Fenton (2013) illustrated that a new ranking system, called the pi-ratings, incorporated in their Bayesian network model, significantly outperforms the ELO ratings. Groll et al. (2018) used a random forest with multiple informative covariates to predict the goals scored by the opposing teams. These estimates where then used as the intensity parameters λ for the Poisson distributions of the opposing teams, and used to derive the outcome class probabilities. Groll et al. (2019) expanded this random forest model to a hybrid random forest model, where first the relative strengths of the competing teams were estimated based on a Poisson maximum likelihood approach. Then secondly these relative strengths were used in a random forest model, combined with other covariates from Groll et al. (2018), to estimate the goals of the opposing teams, from which the outcome class probabilities were again indirectly derived. Baboota and Kaur (2019) used the gradient boosting, naive Bayes, linear support vector machine, RBF support vector machine and random forest algorithm to estimate the outcome distributions directly. The article reports that the gradient boosting algorithm outperformed all other models.

The performance of statistical models is usually only evaluated on a single football league. The following section gives a brief overview of the most popular and important football leagues. Another issue is that different authors use diverse scoring rules to evaluate the performance of their statistical models. After the section about the football leagues, a section dedicated to the various scoring rules is given.

1.3 Football leagues

There are many football leagues throughout the world. Some of the most competitive and popular include the English - Premier League, the German - Bundesliga and the Spanish - La Liga. There are also matches between national teams organised by international football federations, such as the International Federation of Association Football (FIFA), and FIFA confederation such as the Union of European Football Associations (UEFA). These international federations organise football tournaments between nations, usually yearly or over multiple years, e.g. the FIFA World Cup tournament takes place every four years and is the most famous international football championship (Suzuki et al., 2010).

Due to the immense popularity and highly competitive level of the English Premier League, we will include the data from this domestic league in this master dissertation.

1.3.1 English Premier League

The English Premier League is the most famous football league worldwide and is considered to be of the highest competitive level. It consists of 20 teams that play against each other twice a season, for a total of 380 matches. The English Premier League is broadcasted worldwide in 212 countries and reaching 4.7 billion people (Kundu et al., 2019). The revenues generated are therefore enormous and estimated at 2.2 billion euro in television rights and 5.8 billion euro from other sources such as merchandise and ticket sale. These numbers illustrate the magnitude of how successful the English Premier League is.

The highly competitive nature of the English Premier League gives the outcome distribution of matches much randomness. Therefore it is rather challenging to come up with accurate prediction models. One way to measure the randomness of a dataset is by looking at its entropy. An entropy score of 1 means complete randomness. Kundu et al. (2019) reports that for the historical English Premier League data between the seasons of 2005 and 2016, the entropy of the dataset was 0.96, which again supports the assumption that the English Premier League is highly competitive.

1.4 Scoring rules

Scoring rules are functions used to evaluate the performance of predictive models. There is a wide range of scoring rules available, each developed for different purposes and situations. For football match outcomes, which is considered to be an ordinal outcome parameter with three classes, namely home win, draw and away win, there are also numerous choices available. Debate exists over which scoring rule is the most appropriate (Wheatcroft, 2019).

Wheatcroft (2019) considered three properties in scoring rules to be relevant for the evaluations of models that aim to predict football match outcomes. Firstly, a scoring rule must be proper, meaning that it favours predictions that consist of distributions drawn from the actual outcome distribution. Another central property in scoring rules that aim to evaluate football match prediction models, is locality. A scoring rule can either be local or non-local. It is considered local if it only takes the probability from the predicted class into account. Therefore, a non-local scoring rule takes the probability from multiple classes into account. If a score is non-local, a final property to consider is sensitivity to distance. This sensitivity to distance follows the rationale that the predictive outcome classes are ordinal. A scoring rule should, therefore, penalise a model more in case of an observed home win and predicted away win, compared to

a predicted draw (Ley et al., 2019). A scoring rule that is insensitive to distance does not follow the rationale that the predictive outcome classes are ordinal.

Most articles focus on the ranked probability score (RPS), the Brier score (BS) and ignorance score (IGN). Some papers use accuracy to evaluate the performance of their models. However, it is often challenging to predict draws, hence scoring rules that focus on the probability placed on each outcome class, are preferred.

1.4.1 Ranked probability score

Ranked probability score, proposed by Epstein (1969), is considered to be the most appropriate scoring rule by Constantinou and Fenton (2012) and has since gained more recognition. It is currently the most popular and widely used scoring rule for evaluation of football match outcome models. The ranked probability score is a non-local scoring rule that is sensitive to distance. The following function gives the ranked probability score:

$$RPS = \frac{1}{2N} \sum_{n=1}^{N} ((P_{Hn} - y_{Hn})^2 + (P_{An} - y_{An})^2) \tag{1.2}$$

Here the parameters P_{Hm} and P_{Am} are the predicted probabilities of a home win or away win. y_{Hm} and y_{Am} are the observed outcomes, so either 1 or 0. N is the total number of predicted matches.

The ranked probability score is a topic of present debate in the literature. Ever since Constantinou and Fenton (2012) proposed it to be the most appropriate scoring rule, it gained notoriety. However, Wheatcroft (2019) recently published an article where he argued against the importance of the non-locality and sensitivity to distance properties in a scoring rule for the evaluation of models that predict football match outcomes.

The main argument in favour of the ranked probability score is that probabilities placed on outcomes close to the observed outcome should receive a higher reward (Constantinou and Fenton, 2012). If a team is winning by one goal, it takes the opposing team one goal to end the match in a draw and two goals to end in a win for the opposing team. In this regards, the outcome classes are considered ordinal, and thus sensitivity to distance is deemed to be essential in its scoring rule, which makes the ranked probability score an obvious choice. Constantinou and Fenton (2012) then gives hypothetical examples of football matches, from which they conclude that the ranked probability score is favoured, because it assigns the best score to the favoured forecast in each case.

The main counter-argument of Wheatcroft (2019) is that the examples provided by Constantinou and Fenton (2012) are flawed since it compares the performance of the scores under specific outcomes. Instead, the underlying probability distribution of the match must be taken into consideration because the observed outcome gives no information about the actual underlying distribution. Wheatcroft (2019) then reproduced the examples given by Constantinou and Fenton (2012) and shows that the Brier score and in particular the ignorance score are more appropriate scoring rules for football match prediction models.

For a deeper understanding of these examples, we refer the readers to both articles. In this master dissertation, however, we will use a combination of different scoring rules to circumvent this debate.

1.4.2 Brier score

The Brier score (Brier, 1950), or the squared loss function, is quite similar to the ranked probability score but it is insensitive to distance. This insensitivity to distance means that it does not penalize a model more according to the ordinal structure of the outcome classes. The following function gives the Brier score:

$$BS = \frac{1}{N}\sum_{n=1}^{N}\sum_{r=1}^{R}((P_{nr} - y_{nr})^2) \tag{1.3}$$

Here N is the number of predicted matches. The parameter R stands for all of the possible outcome classes. P_{nr} is the probability placed on instance n for outcome class r. Y_{nr} is the observed outcome, for instance n and outcome class r, so either 0 or 1.

1.4.3 Ignorance score

The ignorance score (Gneiting and Raftery, 2007), or the logarithmic loss function, is a scoring rule that is both local and insensitive to distance. The following function gives the ignorance score:

$$IGN = \frac{1}{N}\sum_{n=1}^{N}\left[-\log_2(p(y_n))\right] \tag{1.4}$$

Here $p(y_n)$ is the probability placed on the correct outcome class y of match n. Wheatcroft (2019) found practical evidence that the ignorance score is the most proper scoring rule (Bröcker and Smith, 2007).

1.5 Problem statement

The main problem is the multitude of models that aim to predict football match outcomes on incomparable scales. The first layer of the problem is that scholars use different scoring rules to evaluate the performance of their statistical models. Therefore the comparison between them is challenging to interpret. A second layer is that scholars use data of different leagues. Comparison of models trained on different data is often difficult to understand since different leagues have different competitive levels. A third layer and final layer is that scholars use different training and test protocols for their statistical models.

1.6 Objectives

In this master dissertation, we aim to evaluate the current literature of football outcome prediction models and circumvent the problems mentioned above in section 1.5. The remainder of the master dissertation is built up as follows: section 2 explains the methods used, section 3 shows the results and finally in section 4 the results are discussed.

CHAPTER 2

METHODS

This chapter explains the methods used to create a detailed overview of the models on comparable scales, in detail. First, we will go over the general protocol and data. The following sections explain each evaluated model separately. For each model, first, the method is conceptualised, and if necessary, the data specific to the model are explained.

2.1 Protocol

A uniform protocol brings the evaluated models on comparable scales. The evaluation of each model will use all the scoring rules, mentioned in section 1.4 (ranked probability score, brier score and ignorance score), in conjunction. The models are ranked for each scoring rule, and the best performing model is defined by having the highest average rank. After that, the models are estimated on data originating from the English Premier League, which circumvents the problem of using different leagues.

Most models use vastly different predicting procedures. We aim to bring them all together on a comparable scale. All English Premier League matches in the seasons of 2008 to 2015 are predicted. During the prediction of a season, the previous two seasons of matches are used as data, combined with the first five weeks of the current season, to estimate the model coefficients. The reason for the burn-in period of the first five weeks is to get reliable information on the possible new teams entering the league.

The weeks of a season are predicted in a stepwise manner. After the prediction of each week, the information is added to the data. A football season consists out of 38 weeks, so in total, for each season, 810 matches are used as initial data, and 330 matches are predicted. In total, for the eight seasons, 2640 matches are predicted

Many statistical models from the current literature have some hyperparameters or variables that require estimation. These hyperparameters are not estimated under the new protocol, but merely the same values from the articles are used. If these val-

ues are absent, the parameters will take arbitrary reasonable values as recommended by the literature. We will explicitly mention our reasoning where this occurs.

2.2 Data

The English Premier League data used is from the *engsoccerdata* R package (Curley, 2016). This package is mainly a repository that contains different European football datasets, such as the three English ones (Premier League, FA Cup, Playoff) and also other European leagues (Spain - La Liga, Germany - Bundesliga, Italy - Seria A, Netherlands - Eredivisie).

Date	Season	home	visitor	hgoal	vgoal	result	round
2011-08-13	2011	Fulham	Aston Villa	0	0	D	1
2011-08-13	2011	Liverpool	Sunderland	1	1	D	1
2011-08-13	2011	Newcastle	Arsenal	0	0	D	1
2011-08-13	2011	Wigan	Norwich City	1	1	D	1
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Table 2.1: Example of the data used for the English Premier League (Curley, 2016)

2.3 Models

This section explains the statistical models used in detail. The additional data required by some models is also specified here. The models are ranked to go from more simple approaches to more complex ones.

2.3.1 Naive models

The first two models utilise the available information naively. We will refer to them as the *uniform* and *frequency* models. These models will serve as benchmark instruments since any model that does not outperform them, is rendered ineffective.

Uniform

The uniform model ignores all available data on past football matches and assumes that the probabilities for a home win, away win or draw are uniformly distributed. Effectively this means that for every match, every outcome class gets a probability of $\frac{1}{3}$ assigned to it.

Frequency

Unlike the uniform model, the frequency model does incorporate some information about past matches played. The frequency model estimates that the probabilities for a home win, away win or draw are distributed as the observed frequencies of the home wins, away wins and draws in the past k matches. So effectively the win probability is equal to $p(homewin) = \frac{1}{N}\sum_{k=1}^{N} I(homewin_k)$, analogous we find $p(awaywin) = \frac{1}{N}\sum_{i=k}^{N} I(awaywin_k)$ and $p(draw) = \frac{1}{N}\sum_{i=k}^{N} I(draw_k)$.

2.3.2 Logit regression models

This section explains two articles that both utilise a logit regression approach to predict the outcome classes.

The first article focuses on the ELO rating system to represent the strengths of the opposing teams. The second article uses the plus-minus rating system to calculate the strengths of the individual players relative to the players within the same team and players between opposing teams.

ELO based models

The ELO based models come from the article by Hvattum and Arntzen (2010). Here the ELO rating system, adapted for football matches, is used to estimate the current strengths of the competing teams. An ordered logit regression model then incorporates the ELO ratings as the single covariate.

Hvattum and Arntzen (2010) compared the predictive performance of this model to six other methods, namely the two naive methods equal to the uniform and frequency model mentioned above, two probit regression models derived from Goddard (2005) and two models based on the odds offered by bookmakers. The article reports that the logit regression based on the ELO ratings outperforms the naive and probit regression models from Goddard (2005), but does not outperform the models based on the bookmaker odds.

The ELO rating system, modified for football matches, estimates the current strengths of a team based on historical data. A scoring system is first defined to derive this rating. In this scoring system, a win grants a score of 1, a draw a score of 0.5 and a loss a score of 0. If l_0^i and l_0^j are the current ELO ratings of teams i, the home team, and team j, the away team, respectively, then the ELO rating framework assumes

that, on average, each team should score γ^i and γ^j against each other in a given match. The following formulas give the functions for γ^i and γ^j:

$$\begin{aligned} \gamma^i &= \frac{1}{1 + c^{-(l_0^i - l_0^j)/d}} \\ \gamma^j &= \frac{1}{1 + c^{-(l_0^j - l_0^i)/d}} = 1 - \gamma^i \end{aligned} \tag{2.1}$$

The formulas above are dependent on two parameters, namely c and d, with $c > 1$ and $d > 0$. These parameters are only used to scale the expected scores. Hvattum and Arntzen (2010) reported that $c = 10$ and $d = 400$ is sufficient. Alternative values can give identical rating systems. In 2018, the FIFA rating system was changed to the ELO-based system, with $d = 600$. Alternative methods extend the ELO ratings by incorporating a home advantage effect h. The formula above is then formulated as $\gamma^i = \frac{1}{1+c^{-((l_0^i+h)-l_0^j)/d}}$. For example the website *https://eloratings.net/* uses this formula with $h = 100$.

The scoring system gives the observed scores α^i and α^j for both teams, with $\alpha^i = 1$ if team i won, 0.5 if the match was a draw and 0 otherwise. α^j is calculated as $1 - \alpha^i$. The ELO ratings are updated after every match from l_0 to l_1 by the following formulas:

$$\begin{aligned} l_1^i &= l_0^i + k(\alpha^i - \gamma^i) \\ l_1^j &= l_0^j + k(\alpha^j - \gamma^j) \end{aligned} \tag{2.2}$$

The formulas above are dependent on the parameter k. Unlike the values for the parameters c and d, a more careful value consideration is needed for k. If k is set too low, team ratings take too long to stabilise, and if k is set too high, team ratings will be too volatile. Hvattum and Arntzen (2010) reported that $k = 20$ is reasonable and that after 30 matches the ELO ratings are usually stabilised.

Note that the ELO rating system requires some initial data in order to indicate the current strengths of the opposing teams reliably. For this reason, the protocol mentioned in 2.1 is extended with an initial period of one year, e.g. for the prediction of the season for the year 2008, the season of 2005 is used to get reliable initial estimates of the ELO ratings before starting the training protocol.

Two prediction models are created from this ELO rating system, namely the *basic ELO* and *goal-based ELO*. The basic ELO uses the methodology described above, with $c = 10$, $d = 400$ and $k = 20$. The goal-based ELO extends the basic ELO, by improving the parameter k to depend on the absolute goal difference between the teams. The

incorporation of this goal difference ensures that a higher difference translates to a higher gain in ELO. The parameter k is now extended to $k = k_0(1+\delta)^{\lambda}$, with δ equal to the absolute goal difference. The goal-based ELO thus requires the estimation of four hyperparameters. Hvattum and Arntzen (2010) reported that $c = 10$, $d = 400$, $k_0 = 10$ and $\lambda = 1$ are reasonable.

The ordered logit regression model is then used, with the difference in the ELO ratings of the opposing teams as a single covariate, to make predictions for the match results. The difference in ELO ratings is defined as $x = l_0^i - l_0^j$, with l_0^i equal to the current ELO rating of team i, the home team, and l_0^j for team j, the away team. After each prediction, the ELO ratings are updated. Figure 2.1 shows how these ELO ratings have evolved from 2000 to 2015. The ordered logit regression model is defined by:

$$\begin{aligned} p(y = i|x) &= F(-\Theta_i - \beta x) - F(-\Theta_{i-1} - \beta x) \\ F(z) &= \frac{1}{1+e^{-z}} \end{aligned} \tag{2.3}$$

In the formula above $j \in \{1, 2, 3\}$ represents the outcome classes, with $j = 1$, for an away win, $j = 2$ for a draw and $j = 3$ for a home win. The parameters Θ are used to differentiate between the ordinal values of the dependent variable. Only Θ_1 and Θ_2 need to be estimated, since $\Theta_0 = \infty$ and $\Theta_3 = -\infty$. The function $F(z)$ represents the logit link, with $F(-\infty) = 0$ and $F(\infty) = 1$. So in total the the logit regression needs to estimate three parameters, namely Θ_1, Θ_2 and $\beta_{ELOdiff}$.

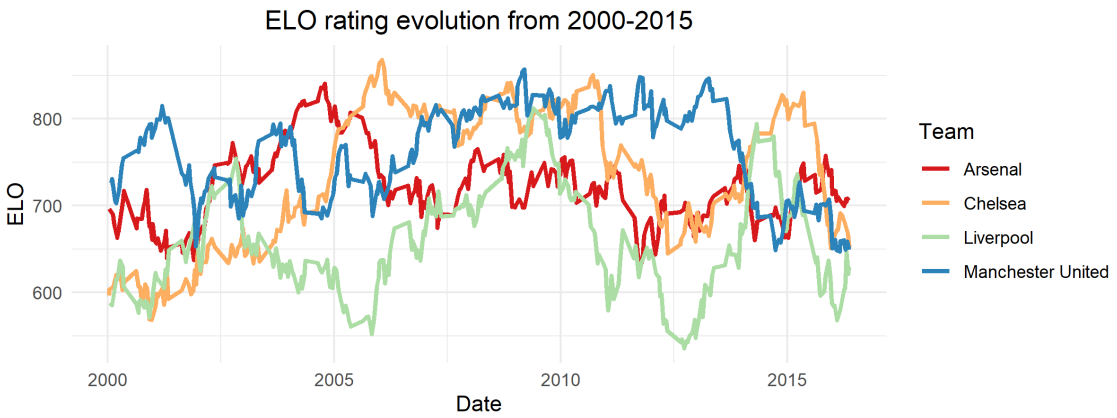

Figure 2.1: Evolution of ELO ratings from 2000-2015

Plus-minus based models

The plus-minus based models are derived from the article by Sæbø and Hvattum (2015). Here the plus-minus rating system, adapted for football matches, is used to estimate the strengths of the players relative to its own teammates and the opposing

team's players. The plus-minus ratings of a team are then calculated as the average plus-minus ratings of the players within a team. Again, an ordered logit regression model, with the difference in the plus-minus ratings of the opposing teams, is used to estimate the outcome class probabilities.

Plus-minus ratings attempt to distribute credit for the goals of a team onto the players responsible for it (Hvattum, 2019). In its simplest form, the plus-minus rating system calculates the goals scored minus the goals conceded for every player during a match. Specifically for football matches, Sæbø and Hvattum (2015), came up with a regularised adjusted plus-minus rating system.

For the regularised adjusted plus-minus ratings, we first need to define segments, within a match, of constant players on the field. Every time a team changes a player on the field, or a player receives a red card, the match is split into two separate segments.

For each segment i, we define an appearance of a player j, by the parameter α_{ij}. The value of α_{ij} is given by the following formula:

$$\alpha_{ij} = \begin{cases} e^{-kt} & \text{If player j plays for the home team in segment i} \\ 0 & \text{If player j does not play in segment i} \\ -e^{-kt} & \text{If player j plays for the away team in segment i} \end{cases} \tag{2.4}$$

In the formulas above, the factor e^{-kt} is dependent on two parameters, namely k and t, and represents a time depreciation effect. The parameter t is the difference between the current time, when the plus-minus ratings need to be estimated, and the time when a match is played, expressed in years. The parameter $k \in [0, 1]$ represents the magnitude of the time depreciation effect. Sæbø and Hvattum (2015) report that $k = 0.2$ is within reason. This value implies that the weight of a match played five years ago contributes only for $\frac{1}{e}(\approx 0.37)$ to the estimation of the plus-minus rating, compared to matches played in the present time.

Since the plus-minus rating system is based on the goals scored minus the goals conceded, we define a parameter β_i. This parameter represents a scaled version of the goals scored minus the goals conceded, in favor of the home team, during a segment i. The following function defines β_i:

$$\beta_i = \frac{90(H_i - A_i)e^{-kt}}{D_i} \tag{2.5}$$

In the formula above, the parameter D_i represents the duration of segment i, $H_i - A_i$ represents goals scored minus the goals conceded in favour of the home team during segment i and e^{-kt} is again the time depreciation effect.

A twelfth dummy player, that is included in every home team's starting lineup, accounts for the home advantage effect. The appearance of the home advantage is thus always equal to e^{-kt}. For the calculation of the plus-minus ratings of a home team, the average of the players and home team advantage must be taken.

Four dismissal dummy variables account for the effect of red cards. When a player j in the home team gets a red card, then the first dismissal dummy gets a value of e^{-kt}, and the player's appearance is changed to 0. A second red card for the home team would lead to a transfer of appearance from the player getting the red card to the second dismissal dummy. Red cards for the opposing teams can nullify the dismissal dummy variables, e.g. if the home team has two red cards and the away team one, then the first dismissal dummy has a value of 0 and the second a value of e^{-kt}.

Finally the plus-minus ratings x are calculated by:

$$x = (a^T a + \lambda I)^{-1} a^T \beta \tag{2.6}$$

The formula above corresponds to the estimation of the coefficients in a penalised linear regression by using the least-squares criterion. The penalisation term is crucial, since many players are joint for most of their playtime. The rating system then struggles to differentiate between them, which causes collinearity issues and inflates the errors. Also, the ratings for players with little playing time are prone to large errors. The penalisation term in equation 2.6 is equivalent to the ridge regression approach and was found by Macdonald (2012) to be the most appropriate penalisation term when estimating plus-minus ratings. Sæbø and Hvattum (2015) report that $\lambda = 3000$ is sufficient.

Note that the plus-minus ratings, just like the ELO ratings, requires some initial data in order to indicate the current strengths of the opposing teams reliably. Arntzen and Hvattum (2020) used the previous five seasons to get those initial ratings, so we will extend the protocol, mentioned in 2.1, with an initial period of five years, e.g., for the prediction of the season of 2015, the seasons from 2008 till 2012 are used to get the initial ratings. Then the seasons 2013 and 2014, in combination with the first five weeks of season 2015, are used to estimate the coefficients of the ordered logit regression.

In this master dissertation, we were not able to incorporate the data for the red cards due to time constraints. Also important to note is that Sæbø and Hvattum (2015) did not use the plus-minus ratings initially to predict football match outcomes, but instead, they used it to evaluate the efficiency of the transfer market. However, in a follow-up study by Arntzen and Hvattum (2020) the difference in plus-minus ratings between the opposing teams, in favour of the home team, is taken as a single covariate in an ordered logit regression, see equation 2.4. Arntzen and Hvattum (2020) also improved the plus-minus rating system to depend on an age effect, the similarity of the players and the competition type.

2.3.3 Poisson based models

The third set of models comes from the article by Ley et al. (2019). Here eight different statistical models are compared in their performance to predict football match outcomes. From those models, the independent and bivariate Poisson had the highest predictive power. Additionally, these models create a ranking method based on a maximum likelihood approach. We will refer to this ranking parameter as the ability of a team. Figure 2.2 shows how these have evolved between 2000 and 2015.

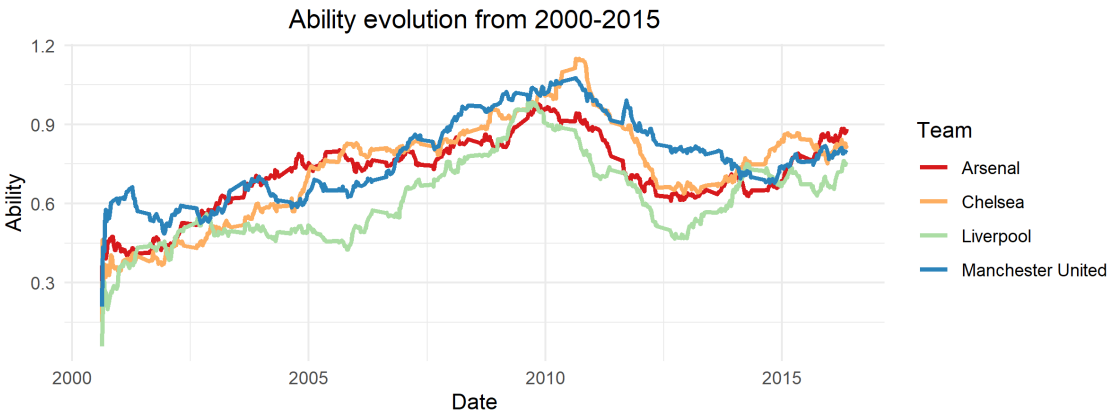

Figure 2.2: Evolution of the abilities from 2000-2015

Independent Poisson

The independent Poisson models the goals of the opposing teams i and j by the random variables $G_{i,m}$ and $G_{j,m}$ for match m. These random variables follow a Poisson distribution. The following formulas give the joint density distribution of observing x goals for team i and y goals for team j, under the assumption that both random variables are independent from each other:

$$P(G_{i,m} = x, G_{j,m} = y) = \frac{\lambda_{i,m}^{x}}{x!} exp(-\lambda_{i,m}). \frac{\lambda_{j,m}^{y}}{y!} exp(-\lambda_{j,m}) \tag{2.7}$$

The parameters $\lambda_{i,m}$ and $\lambda_{j,m}$ are the expected goals scored by team i and j, respectively. These λ's are estimated to represent the abilities of the opposing teams. The following formulas demonstrate this:

$$\begin{aligned} \lambda_{i,m} &= exp(c + (r_i + h) - rj) \\ \lambda_{j,m} &= exp(c + rj - (r_i + h)) \end{aligned} \tag{2.8}$$

Here h stands for the home effect, c is the intercept, r_i and r_j are the relative abilities of team i, the home team, and team j, the away team, respectively. The ability parameters, intercept and home team advantage are estimated by a maximum likelihood approach which takes the following formula:

$$L = \prod_{m=1}^{M} \prod_{i,j \in (1,\ldots,T)} \left(\frac{\lambda_{i,m}^{g_{i,m}}}{g_{i,m}!} exp(-\lambda_{i,m}). \frac{\lambda_{j,m}^{g_{j,m}}}{g_{j,m}!} exp(-\lambda_{j,m}) \right)^{y_{ijm}.w_{time,m}} \tag{2.9}$$

In the formula above, the parameter $m \in \{0, \ldots, M\}$ is an index for the match, and the parameter T encompasses all the different teams. The random variable y_{ijm} is equal to 1, if i and j stand for the home and away team, respectively, in match m, else y_{ijm} is equal to 0. The parameters $g_{i,m}$ and $g_{j,m}$ are the actual observed goals scored by each team. $W_{time,m}$ serves as a decay function to reflect a smooth time depreciation effect. This function is given by $W_{time,m}(x_m) = \frac{1}{2}^{\frac{x_m}{Halfperiod}}$, which implies that a match played *Half period* days ago contributes half as much to the likelihood function compared to matches played in the present time.

Ley et al. (2019) reports that the optimal *Half period* was 360 days for the *Independent Poisson* model and 390 days for the *Bivariate Poisson* model for the data coming from the English Premier League.

Bivariate Poisson

Ley et al. (2019) extended the basic independent Poisson by adding a direct correlation coefficient between the scores, based on the bivariate Poisson model suggested by Karlis and Ntzoufras (2003). The goals of the opposing teams i and j are now modelled by the random variables $G_{i,m} = X_{i,m} + X_C$ and $G_{j,m} = X_{j,m} + X_C$ for match m, where $X_{j,m}$, $X_{i,m}$ and X_C follow a Poisson distributed with the respective intensity parameters $\lambda_{j,m}$, $\lambda_{i,m}$ and λ_C. The expected goals, $E(G_{i,m}) = \lambda_{i,m} + \lambda_C$ and $E(G_{j,m}) = \lambda_{j,m} + \lambda_C$

, scored by team i and j respectively, now take the correlation term into account. The parameter λ_C is the correlation between the scores of both opposing teams. The $\lambda_{i,m}$ and $\lambda_{j,m}$ parameters are still estimated to represent the abilities of the opposing teams, with the same equation as 2.8. The following formula gives the joint density distribution of the bivariate Poisson:

$$P(G_{i,m} = x, G_{j,m} = y) = \frac{\lambda_{i,m}^{x}\lambda_{j,m}^{y}}{x!y!} exp(-(\lambda_{i,m} + \lambda_{j,m} + \lambda_C)) \sum_{k=0}^{min(x,y)} \binom{x}{k}\binom{y}{k} k! \left(\frac{\lambda_C}{\lambda_{i,m}\lambda_{j,m}}\right)^k \quad (2.10)$$

The next formula gives the appropriate likelihood function:

$$L = \prod_{m=1}^{M} \prod_{i,j\in(1,\ldots,T)} \left(\frac{\lambda_{i,m}^{g_{i,m}}\lambda_{j,m}^{g_{j,m}}}{g_{i,m}!g_{j,m}!} exp(-(\lambda_{i,m} + \lambda_{j,m} + \lambda_C)) \sum_{k=0}^{min(g_{i,m},g_{j,m})} \binom{g_{i,m}}{k}\binom{g_{j,m}}{k} k! \left(\frac{\lambda_C}{\lambda_{i,m}\lambda_{j,m}}\right)^k \right)^{y_{ijm}.w_{time,m}} \quad (2.11)$$

.

The parameters of the likelihood function are interpreted in the same way as in equation 2.9. If i and j stand for the home and away team, y_{ijm} is still equal to 1 and else to 0. The observed goals are denoted as $g_{i,m}$ and $g_{j,m}$. $W_{time,m}$ is still the weight function for the time decay effect.

2.3.4 Weibull count model

The fourth model comes from the article by Boshnakov et al. (2017). Just like the Poisson based models, the fourth approach is also goal-based, which means that it models the goal distributions of the opposing teams. However, unlike the Poisson based models, the fourth approach assumes that the goals now follow a Weibull count distribution. The following formula gives the Weibull count density distribution:

$$\begin{aligned} p(X(t) = x) &= \sum_{j=x}^{\infty} \frac{(-1)^{x+j}(\lambda t^c)^j \alpha_j^x}{\Gamma(cj+1)} \\ \Gamma(z) &= \int_0^{\infty} t^{z-1}e^{-t}dt \end{aligned} \quad (2.12)$$

In the function above $\alpha_j^0 = \Gamma(cj+1)/\Gamma(j+1)$, for $j = 0, 1, 2, \ldots$, and $\alpha_j^{x+1} = \sum_{m=x}^{j-1} \alpha_m^x \Gamma(cj - cm + 1)/\Gamma(j - m + 1)$, for $x = 0, 1, 2, \ldots$ and $j = x + 1, x + 2, x + 3, \ldots$. The parameter λ can be seen as the scoring rate per match, which is comparable to the λ used in the Poisson distribution. The c parameter is the shape of the distribution and allows

for extra flexibility. The dispersion of the Weibull count is denoted by the hazard $h(t) = \lambda c t^{c-1}$. Note that if $c = 1$, the Weibull count distribution is equal to a Poisson distribution, since the dispersion is then equal to λ. If $c > 1$ the distribution is over-dispersed and if $c < 1$ the distribution is under-dispersed. For the estimation of this density function, we used the R package *Countr* (Baker et al., 2016). Figure 2.3 illustrates the differences between the Poisson and Weibull count distribution, fitted to the goals of the home and away teams in the English Premier League, and clearly shows that the Weibull count distribution has a better fit, in particular for lower values of the observed goal values.

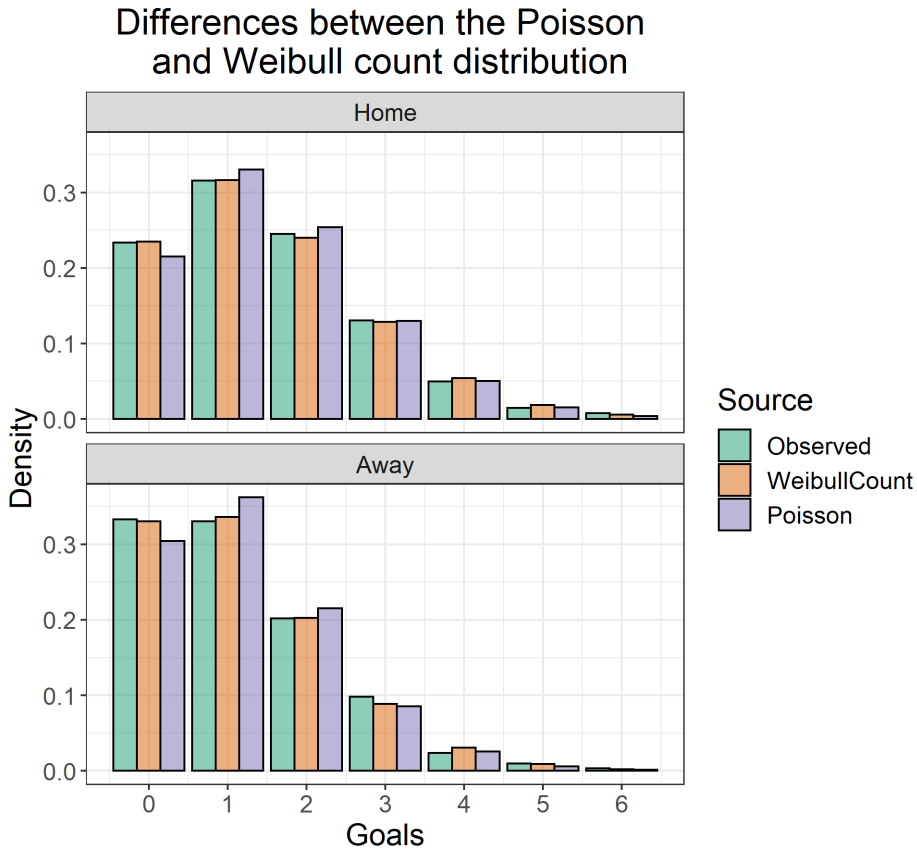

Figure 2.3: Difference between Weibull Count and Poisson distribution

Boshnakov et al. (2017) used the Weibull count distribution with a copula dependence between the goal distributions of the home and away teams to create a bivariate prediction model for the outcomes of football matches. A copula C is a multivariate distribution, for which the marginal distributions are uniform between $[0, 1]$. Here a copula C is used to glue the marginal cumulative distributions, of the home and away goals, together. The following formula illustrates this:

$$C(F_1(y_1), F_2(y_2)) = F(y_1, y_2) \tag{2.13}$$

Equation 2.14 shows how a copula C can be used to glue the marginal cumulative distributions, $F_1(y_1)$ and $F_2(y_2)$, together from the joint density distribution $F(y_1, y_2)$. Boshnakov et al. (2017) reports that a Frank's copula provided the best fit to the data, and will thus be the copula of choice. The formula below gives the equation for Frank's copula:

$$C(u, v) = -\frac{1}{k} \log\left(1 + \frac{(e^{-ku} - 1)(e^{-kv} - 1)}{e^{-k} - 1}\right) \tag{2.14}$$

In this formula, the parameter k is the dependence between u and v, which in our case are the marginal cumulative distributions. The coefficients of the bivariate Weibull count model are then estimated by using a maximum likelihood approach. The maximum likelihood takes the following formula:

$$\begin{aligned} L(k, \alpha, \beta, c) = \prod_{k \in (k:t_k < t)}^{M} & (C(F_1(y_{1i}), F_2(y_{2i})) - C(F_1(y_{1i} - 1), F_2(y_{2i})) \\ & - C(F_1(y_{1i}), F_2(y_{2i} - 1)) + C(F_1(y_{1i} - 1), F_2(y_{2i} - 1)) * e^{-\epsilon(t - tk)} \end{aligned} \tag{2.15}$$

Here F_1 and F_2 are again the cumulative Weibull count distributions for the home and away teams respectively. The cumulative Weibull count distributions require two variables, the rate parameter λ and shape parameter c. For every home team the parameter λ is defined as: $\log(\lambda_i) = \alpha_i + \beta_i + \gamma$, with α_i equal to the attack strength of team i, β_i equal to the defence strength and γ the effect of playing as the home team. For every away team the parameter λ is defined as: $\log(\lambda_j) = \alpha_j + \beta_j$. The shape parameters c_h and c_a, for the home and away team respectively, are expected to be constant. The parameters y_{1i} and y_{2i} are the observed goals.

The maximum likelihood procedure thus has to estimate $2T + 4$ parameters, namely α's and β's for all the teams T, c_h and c_a which are the shape parameters of the home and away team, h the home team effect and k the dependence term from Frank's copula.

Finally, the last unknown parameter ϵ models the time depreciation effect. In formula 2.16, the term $t - t_k$ stands for the difference between the time t_k, when a historical match m was played, and the current time t, expressed as number of days. The value for ϵ is obtained by maximizing the next function:

$$T(\epsilon) = \sum_{k=1}^{N} (\delta_k^H \log p_k^H + \delta_k^A \log p_k^A + \delta_k^D \log p_k^D + \gamma_k^{O2.5} \log p_k^{O2.5} + \gamma_k^{U2.5} \log p_k^{U2.5}) \tag{2.16}$$

In this formula, $\delta_k^H = 1$ if the home team won the match k, δ_k^A and δ_k^D are interpreted analogously. p_k^H, p_k^A and p_k^D are the maximum likelihood estimates for the probability of a home win, home loss and draw, respectively, in match k. The parameter $\gamma^{O2.5} = 1$, if there are more than 2.5 goals in match k, and $\gamma^{U2.5} = 1$ if there are fewer than 2.5 goals. $p_k^{O2.5}$ and $p_k^{U2.5}$ are the maximum likelihood estimates for the probability of observing more or fewer than 2.5 goals in match k. Boshnakov et al. (2017) reports that a value of $\epsilon = 0.002$ is reasonable. This value implies that the weight of a match played 500 days ago contributes only for $\frac{1}{e}(\approx 0.37)$ to the maximum likelihood estimation, compared to matches played in the present time.

2.3.5 Machine learning models

The next models come from the article by Baboota and Kaur (2019). In this article, Baboota and Kaur (2019) compared five different machine learning approaches in their ability to predict football match outcomes. The considered models are the naive Bayes, linear support vector machine, RBF support vector machine, random forest and gradient boosting. From those five models, the random forest and gradient boosting outperform the other models and will thus be included in this master dissertation. Firstly, we will go over the specific data used for these models.

Data

The models used by Baboota and Kaur (2019) use a set of highly informative engineered features. The first set feature involves the different FIFA-ratings of the opposing teams, namely the attack, midfield, defence and overall rating. The EA Sports company constructed these ratings to be used in the FIFA game series. The ratings can be scraped from the FIFA index database (https://www.fifaindex.com/). The statistical model then uses the difference between the respective home and away team ratings, given by the following formula, $R_i = R_i^H - R_i^A$, for $R_i \in$ (attack, midfield, defence, overall), and R^H and R^A, standing for the home and away team respectively. This trend, of using the difference between the home and away team, can be assumed for all the following continuous features.

The next feature is the goal difference. This feature is a sum of the goal differences from the preceding matches. The goal differences for the kth match is given by $GD = \sum_{j=1}^{k-1} GS_j - \sum_{j=1}^{k-1} GC_j$, where GS and GC stand respectively for the goals scored and conceded.

The third set of features incorporate information of a team's recent performance. Specifically, the feature contains the average the number of corners, shots on target

and goals of the past k matches. the formula for the jth match is given by, $\mu_j^i = (\sum_{p=j-k}^{j-1} \mu_p^i)/k$, with $\mu^i \in (Corners, Shotsontarget, Goals)$. The hyperparameter k requires tuning. Baboota and Kaur (2019) report that $k = 6$ is optimal for the random forest and gradient boosted model. The data for this feature can be obtained from the Football UK website (https://www.bbc.com/sport/football/).

The fourth set of features are also engineered to represent the recent performance of a team. Namely two indicators, the streak and weighted streak, were used for this purpose. The streak is meant to capture the recent increase/decrease in performance of a team. The streak is calculated by giving a score to each match result, and then the mean score of the k preceding matches is taken. Note that k is the same hyperparameter as mentioned above. The scores follow the 3-1-0 rule. A win grants a score of 3, a draw a score of 1 and a loss a score of 0. The weighted streak is calculated by updating the streak with a time depreciation effect. The oldest observation (j-k) gets a value 1, and the most recent observation (j-1) gets a value k. The following formulas give the streak (δ) and weighted streak (ω) for the jth match:

$$\delta_j = \left(\sum_{p=j-k}^{j-1} res_p \right) / 3k$$
$$\omega_j = \sum_{p=j-k}^{j-1} 2 \frac{p-(j-k-1)res_p}{3k(k+1)} \tag{2.17}$$

Here $res_p \in (0, 1, 3)$ stands for the score given to the outcome of the match. The term $3k$ ensures that the streak is normalised between 0 and 1. In the weighted streak formula, the term $p-(j-k-1)$ ensures that the oldest observation (j-k) gets a value 1, and the most recent observation (j-1) gets a value k, and the term $3k(k+1)/2$ now ensures the normalisation.

The last feature, named form, aims to display the performance of a team during an individual match. Just like the streak feature, a team's form gives information about the recent performance of a team. However, contrary to the streak feature, the form feature displays a team's performance relative to the opposing team. After every match, the form values of a team are updated. The form feature rewards a team with low form, defeating a team with high form, more, and vice-versa. If the match ends in a draw, the form of the weaker team increases, while the form of the stronger team decreases. The initial value of the form of each team is 1, and updated after every match by the following formula:

In the case of a win or loss:

$$\begin{aligned} \epsilon_j^\alpha &= \epsilon_{j-1}^\alpha + \gamma \epsilon_{j-1}^\beta \\ \epsilon_j^\beta &= \epsilon_{j-1}^\beta - \gamma \epsilon_{j-1}^\alpha \end{aligned} \tag{2.18}$$

In the case of draw:

$$\begin{aligned} \epsilon_j^\alpha &= \epsilon_{j-1}^\alpha - \gamma \left(\epsilon_{j-1}^\alpha - \epsilon_{j-1}^\beta \right) \\ \epsilon_j^\beta &= \epsilon_{j-1}^\beta - \gamma \left(\epsilon_{j-1}^\beta - \epsilon_{j-1}^\alpha \right) \end{aligned} \tag{2.19}$$

With ϵ_j^α and ϵ_j^β equal to the form of team α and team β respectively, in the jth match. The parameter γ is referred to as the stealing fraction, e.g. if team α wins over team β, it steals a fraction γ of team β's form. If the match ends in a draw, the weaker team gets a positive update and the stronger team a negative update, proportional to the difference in their respective forms. Baboota and Kaur (2019) report that the value of the stealing fraction γ is equal to 0.33.

Feature	Equation
Form	$\epsilon_j^\alpha - \epsilon_j^\beta$
Streak	$\delta_j^\alpha - \delta_j^\beta$
Weighted streak	$\omega_j^\alpha - \omega_j^\beta$
Corners	$\mu_{corners,j}^\alpha - \mu_{corners,j}^\beta$
Goals	$\mu_{goals,j}^\alpha - \mu_{goals,j}^\beta$
Shots on target	$\mu_{shots,j}^\alpha - \mu_{shots,j}^\beta$
Goal difference	$\mathrm{GD}_j^\alpha - GD_j^\beta$
FIFA_{attack}	$\mathrm{FIFA}_{attack,j}^\alpha - FIFA_{attack,j}^\beta$
$\mathrm{FIFA}_{midfield}$	$\mathrm{FIFA}_{midfield,j}^\alpha - FIFA_{midfield,j}^\beta$
$\mathrm{FIFA}_{defence}$	$\mathrm{FIFA}_{defence,j}^\alpha - FIFA_{defence,j}^\beta$
$\mathrm{FIFA}_{overall}$	$\mathrm{FIFA}_{overall,j}^\alpha - FIFA_{overall,j}^\beta$

Table 2.2: Features used in (Baboota and Kaur, 2019)

Note that many of the variables are only calculated after $k = 6$ weeks. Thus a burn-in period of 6 weeks is needed to get reliable estimates of the features. Baboota and Kaur (2019) reports that the highest performance is found by the random forest and gradient boosting algorithm. The following subsections give some detail about these models.

Random Forest

Random forest models, first proposed by Breiman (2001), is a method that ensembles decision trees. These decision trees are used for regression or classification

problems. The decision trees work by partitioning the predictor space (X_1, ..., X_p) repeatedly into multiple, non-overlapping, distinct regions (R_1, ..., R_j). Usually, this is done with binary splits, intending to find homogeneous response values within the same region and heterogeneous response values between them (Groll et al., 2019). Figure 2.4 illustrates how this progress works with a dendrogram. The decisions trees predict values by averaging over the response values, for regression trees, or using the majority vote, for classification approaches. On figure 2.4, we observe four different splits, that define five distinct regions. However, many more splits could be used, which makes decision trees prone to overfitting.

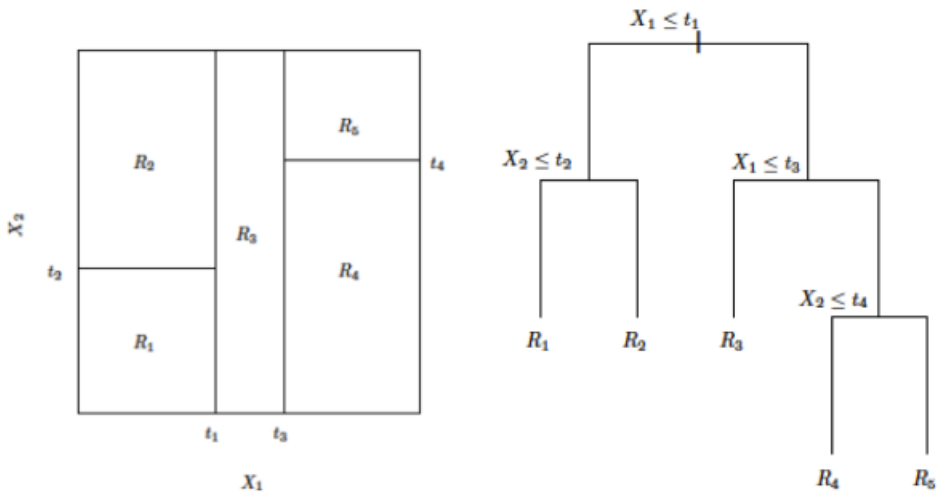

Figure 2.4: Example of decision trees. source: Ley, C. (2020). Big data science, Chapter 4 - Tree-based method [PowerPoint slides], course material University Ghent.

Random forests circumvent this issue of overfitting by using a bagging approach. This bagging approach first creates multiple bootstrapped datasets, and for each such dataset, a decision tree is created. The random forest model aggregates the predictions over the multiple individual decision trees. Combining all of these individual decision trees has the advantage of making the predictions unbiased and reducing the variance among them. Another improvement of random forests over decision trees is that random forest only uses a random subsample of the original predictor space. This reduces the correlation between the multiple decision trees over different bootstrapped datasets. The selection of the random subset of predictors is usually done in two steps. First, the decision trees only use a random subset of original predictor space, and secondly, at every node, a random subset of the predictor space of the decision tree is used to find the best split.

We use the programming language Python version 3.7.6 with the machine learning software sklearn to create the random forest classifier. The table below shows the specific hyperparameters used.

The first hyperparameter, name *criterion*, measures the quality of a split. It measures the amount of entropy or impurity that each feature removes when it is used in a node. The second hyperparameter, named *max features*, gives the maximum number of features that can be randomly selected at every node to find the best split. A value of log2 indicates that if the predictor space of the decision tree has eight features, three randomly selected features can be used in every node. The third hyperparameter, *min samples leaf*, indicate the minimum number of samples needed to form a distinct region (or leaf). A value of two would mean that a split will only occur if the distinct regions, that are formed after the splitting, each have at least two samples in them. The fourth hyperparameter, *min sample split*, is similar. A value of 100 indicates that 100 samples are needed to split an internal node. Both the min samples leaf and min samples split are used to smooth out the data and attempt to reduce the probability of learning noise. The last hyperparameter, *N estimators*, indicates the number of bootstrapped datasets, with their respective decision trees, that are aggregated into a final random forest. Usually, the value for this hyperparameter is set high enough, so that the predictions obtain the feature of being unbiased, and low enough to reduce computational demands.

Hyperparameter	Description	Value used
Criterion	The function used to measure the quality of a split	gini
Max features	The maximum number of features for the best split	log2
Min samples leaf	The minimum number of samples at region	2
Min samples split	The minimum number of samples needed to split an internal node	100
N estimators	The number of independent trees in the forest	150

Table 2.3: Hyperparameters used in the random forest of (Baboota and Kaur, 2019)

Gradient boosting

Just like the random forest, gradient boosting is a technique that ensembles decision trees. However, unlike the random forest, it does not use a bagging approach, but alternatively, it uses the boosting principle. The difference between a bagging and a boosting approach is that in a boosting approach, the decision trees are not trained independently from each other. Instead, the decision trees are trained in sequence on the entire data set. The model's accuracy increases with each iteration.

We again use the programming language Python version 3.7.6 with the open-source software XgBoost to create the gradient boosted classifier (Chen and Guestrin, 2016). The table below shows the specific hyperparameters used.

The first hyperparameter, *gamma*, is the minimum loss reduction required to make a further partition on an internal node of the tree. A value of 0.5 means that the loss needs to be reduced by 0.5 before splitting a node. A large value of gamma means that the algorithm will be more conservative. The second hyperparameter is the *learning rate*. In gradient boosting algorithms, trees are added to the model sequentially. The newly added trees aim to correct the residual errors in the older trees, which can cause the model to overfit. The learning rate aims to shrink the weight of these new trees. Lower values for the learning rate means that the boosting algorithm stays more conservative. The third hyperparameter, *max depth*, indicates the maximum depth of a tree. A value of three means that only three consecutive splits can be used for each tree. Larger values for this hyperparameter would make the model more complex and likely to overfit. The final hyperparameter, *lambda*, is a ridge regularization term on weights. It is used to reduce the probability of overfitting, and a large value will make the model more conservative.

Hyperparameter	Description	Value used
Gamma	The minimum loss reduction required to make a split	0.5
Learning rate	Boosting learning rate	0.025
Max depth	Maximum depth per tree	3
Lambda	Controls the regularization strength	0.6

Table 2.4: Hyperparameters used in the random forest of (Baboota and Kaur, 2019)

2.3.6 Hybrid random forest model

The final model, proposed by Groll et al. (2019), uses a hybrid modelling approach. This hybrid approach combines both the bivariate Poisson, from section 2.3.3, and a random forest regression model.

Many covariates are gathered and concatenated into a single, highly informative dataset. One of those covariates are the ability parameters estimated by the maximum likelihood approach of the bivariate Poisson, see section 2.3.3 and figure 2.2. The random forest approach then uses all these covariates to model the goals scored by both the home and away team. The Skellam distribution, see section 1.2.1, uses these estimated goals as intensity parameters, λ_1 and λ_2, to derive the probabilities for each outcome class.

Figure 3.2, which comes from the original article by Groll et al. (2019), shows that the ability parameter is highly informative and essential for the performance of the model. The implementation of this model was done in the programming language R version 3.6.1 by using the *ctree* function from the *party* package. The package *mlr* is used to specify the hyperparameters of the hybrid random forest model. As mentioned in section 2.3.5, the random forest model uses a bagging approach, which bootstraps

the data and creates independent decision trees on each of them. If the number of trees used is large enough, the predictions are considered to be unbiased, and the variance is greatly reduced. Predictions are made by averaging over the predictions from the individual trees. The hyperparameter B, for the number of trees, does not require tuning as long as it is sufficiently large, so it gets the feature of unbiasedness (Probst and Boulesteix, 2017). Groll et al. (2019) reports that the number of trees, $B = 500$, is sufficient.

Data

We will now go over all the covariates used in the dataset from the original article by Groll et al. (2019). The dataset used in the original article was created for the national teams. Unfortunately, due to time constraints, we were not able to reproduce the complete dataset for the English Premier League.

Variable	Description
Sportive variables	
ODDSET probability	The bookmaker odds converted to winning probabilities
FIFA rank	The FIFA rank of the national teams based on the last four years
ELO rating	The ELO rating of the national teams
ABILITIES	The ability of a team estimated by the bivariate Poisson model, see 2.3.3
Economic variable	
GDP per capita	The increase in GDP between 2002-2004 for every nation
Population	The nation's population size
Team structure variables	
Host	Dummy indicating if a national team is the hosting country
Continent	Dummy indicating if the national team is on the same continent as the host
Confederation	A categorical variable indicating the confederation of a team
Home advantage variables	
Maximum teammates	The maximum number of team mates for each squad
Second maximum teammates	The second maximum number of team mates for each squad
Age	The average age of each squad
Champion league players	Number of Champion league players in a squad
Legionnaires	Number of players in a squad, playing in clubs abroad
Coach variables	
Age	The age of a team's coach
Tenure	The tenure of a team's coach
Nationality	Dummy indicating if a coach has the same nationality as the team

Table 2.5: Variables used in the hybrid random forest of Groll et al. (2019)

Since we were not able to replicate all the essential features of the dataset during the timeframe of this master dissertation, we decided to focus on a subset of the most informative ones based on figure 3.2, which comes from the original article by Groll et al. (2019).

First, all of the sportive variables were included since those were the most informative for the hybrid random forest model. The ability variable is by far the most important feature. This variable was replicated by using the method described in the bivariate Poisson, section 2.3.3. The second most informative feature are the ELO ratings of

the teams. These were obtained through the goal-based ELO model from Hvattum and Arntzen (2010), section 2.3.2. For the FIFA rank data, we used the overall FIFA rank from the data set of Baboota and Kaur (2019), section 2.3.5. The odds data were obtained from the website *http://www.football-data.co.uk/* and specifically, we used the odds from the bet365 company.

From the home advantage variables, only the age variable was used, since all others were deemed irrelevant for the English Premier League. The age variable, which is the average age of the players in a team, was scraped from the website *https://www.worldfootball.net/* . On that website, the dates of birth of all the English Premier League players, given a season, were listed. Through their dates of birth and the current season, their age could be reverse-engineered.

Most of the economic and team structure variables were not relevant for the English Premier League, e.g. the GDP per capita would be the same for every team and were thus not included. The data concerning the coaches variables were not included due to time constraints. These variables were only mildly informative for the hybrid random forest model, so no significant performance loss is expected.

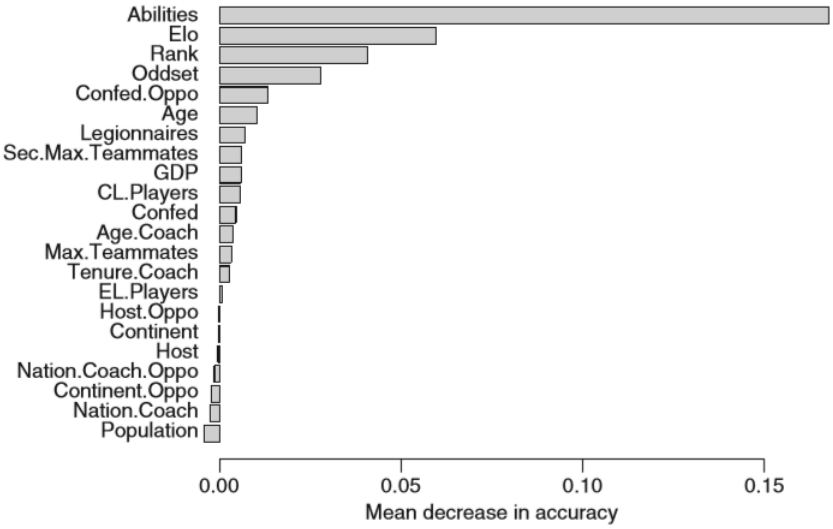

Figure 2.5: Barplot of the variable importance in the original hybrid random forest of Groll et al. (2019)

We also incorporate the random forest model, discussed in Groll et al. (2018), which is identical to the hybrid random forest described above, but without incorporating the information of the abilities of the teams. In order to differentiate between the random forest of Baboota and Kaur (2019) and Groll et al. (2018), we will refer to both models as Random forest$_1$ and Random forest$_2$ respectively. The reason for incorporating the basic random forest is to illustrate how much predictive power is improved by using the hybrid approach.

CHAPTER 3

RESULTS

This chapter will discuss the results of this master dissertation. The scoring rules, mentioned in section 1.4, evaluate and rank the models per scoring rule. The average rank then defines the overall best performing model. After that, we will go over the specific results for each model separately and reference the values for the scoring rules reported in the original articles.

3.1 General results

Table 3.1 shows the average scores (standard deviation) for all the models per scoring rule. Lower scores translate to a better predictive performance of the model. This table shows that models with more sophisticated techniques performed better on average compared to more simplistic models. The models from Baboota and Kaur (2019) significantly outperform the other models, with the gradient boosting model the best among them. Nonetheless, the most complex model, the hybrid random forest, only mildly outperforms the goal-based ELO model, which is considered to use a more simplistic approach.

Another example of this is the Weibull count model, which, despite having a better fit to the observed goals, performs worse than the Poisson-based alternatives. The Poisson-based and ELO models are very similar in their performance. More parsimonious models and models with fewer assumptions are seemingly better at predicting football match outcomes.

Model	Ignorance score	Brier score	RPS Score	Mean rank
Gradient boost	0.933 (0.361)	0.353 (0.175)	0.132 (0.073)	1.000
Random forest$_1$	0.964 (0.350)	0.363 (0.172)	0.141 (0.072)	2.000
Hybrid random forest	1.387 (0.729)	0.571 (0.353)	0.190 (0.135)	3.000
Random forest$_2$	1.408 (0.723)	0.581 (0.350)	0.195 (0.136)	4.000
Goal-based ELO	1.409 (0.752)	0.581 (0.359)	0.197 (0.147)	5.000
Bivariate Poisson	1.417 (0.801)	0.583 (0.355)	0.198 (0.145)	6.333
Basic ELO	1.415 (0.749)	0.584 (0.357)	0.198 (0.147)	6.667
Independent Poisson	1.421 (0.762)	0.585 (0.355)	0.198 (0.141)	8.000
Plus-minus	1.460 (0.759)	0.605 (0.356)	0.207 (0.152)	9.333
Weibull count	1.464 (0.526)	0.610 (0.261)	0.203 (0.101)	9.667
Frequency	1.533 (0.395)	0.641 (0.193)	0.225 (0.089)	11.000
Uniform	1.585 (0.000)	0.667 (0.000)	0.235 (0.073)	12.000

Table 3.1: The mean (sd) score for each model per scoring rule and the average rank.
Random forest$_1$ from Baboota and Kaur (2019) and Random forest$_2$ from Groll et al. (2018)

3.2 ELO based models

For the basic ELO, Hvattum and Arntzen (2010) reported a Brier score with a mean (standard deviation) of $0.627(0.248)$ and for the goal-based ELO a Brier score of $0.626(0.249)$. For these models, we respectively report brier scores of $0.584(0.357)$ and $0.581(0.359)$. Hvattum and Arntzen (2010) reports an ignorance score of $1.502(0.503)$ and $1.499(0.506)$ for the basic ELO and goal-based ELO, respectively. We report ignorance scores of $1.415(0.749)$ and $1.409(0.752)$. Our findings are likely to be better due to the burn-in of the first five weeks, which excluded these observations from the testing procedure. The reported scores are still within each other's standard deviations. Note that the deviations we report are from the individual observations to the mean, not the standard deviation of the mean, which we would specifically denote as the standard error.

The standard deviations reported in the original paper by Hvattum and Arntzen (2010) are lower compared to our findings. This might be due to the differences in the training protocol. Hvattum and Arntzen (2010) used two seasons to get initial ELO ratings, then five seasons were used to estimate the coefficients of the ordered logit regression, and finally, eight seasons were used to test on. The coefficients of the ordered logit regression from Hvattum and Arntzen (2010) were also updated after the prediction of a match.

Table 3.2 shows the coefficients of the ordered logit regression, estimated in this master dissertation, by the end of the season 2015.

Parameter	Value	Standard error	t-value
β_{ELO}	0.006	0.001	12.310
$\Theta_{A\|D}$	-0.912	0.068	-13.328
$\Theta_{D\|H}$	0.238	0.063	3.784

Table 3.2: Coefficients for the ELO ordered logit regression by the end of season 2015

3.3 Plus minus ratings

For the ordered logit regression based on the plus-minus ratings, Arntzen and Hvattum (2020) report a brier score of 0.614 and an ignorance score of 1.476. Our results report a brier score of $0.605(0.356)$ and an ignorance score of $1.460(0.759)$, which are thus within each other's magnitude. Surprisingly, we only included a more simplistic version of the plus-minus ratings, but the predictive performance does not seem to suffer. We have to note, however, that the original paper does not use a burn-in period.

Table 3.3 shows the coefficients of the ordered logit regression, estimated in this master dissertation, by the end of the season 2015.

Parameter	Value	Std. Error	t-value
β_{PM}	45.68	4.04	11.31
$\Theta_{A\|D}$	0.43	0.14	3.20
$\Theta_{D\|H}$	1.63	0.15	10.89

Table 3.3: Coefficients for the plus-minus ordered logit regression by the end of season 2015

3.4 Poisson based models

For the independent and bivariate Poisson, Ley et al. (2019) reports a ranked probability score of 0.1954 and 0.1953, respectively. Our findings are very similar, $0.1983(0.1407)$ and $0.1975(0.1452)$ for the independent and bivariate Poisson, respectively. The original article does not report any standard deviations, but the scores are well within the range of our observed findings. The protocol used by Ley et al. (2019) is identical to the protocol used in this master dissertation. First, the past two years and the first five weeks of the current season were used as initial data, and the data was updated after the prediction of a week. The only difference is that Ley et al. (2019) predicted from season 2008 till 2017, while we predicted from the season of 2008 till 2015.

3.5 Weibull count

The article for the Weibull count by Boshnakov et al. (2017) does not incorporate any of the scoring rules from section 1.4. Instead, it uses the Akaike information criterion (AIC) to compare the copula-based Weibull count model to a copula Poisson model, an independent Weibull count model and independent Poisson model. The lowest AIC was reported for the copula Weibull count model. Unfortunately, we were not able to confirm these findings, since the Weibull count did not outperform the independent Poisson.

3.6 Machine learning models

For the machine learning models, by Baboota and Kaur (2019), the article reports a ranked probability score of 0.2100 and 0.2137 for the gradient boosting model and random forest model respectively. However, Baboota and Kaur (2019) also include figure 3.3 in their article and from that figure, we can indicate that the ranked probability scores of both the gradient boosting and random forest are around 0.173. This information seems inconsistent in the original article. Our finding reports a much lower ranked probability score of $0.132(0.073)$ and $0.141(0.072)$, for the gradient boosting model and random forest model, respectively. In the original article, the matches are not estimated in a stepwise manner. The matches from 2005 to 2014 were used as training data, and the matches from 2015 to 2016 were used as test data. The stepwise addition of information might explain why our findings report lower ranked probability scores. Figure 3.1 shows the feature importance for both models.

3.7 Hybrid model

Finally, the article of the hybrid random forest, by Groll et al. (2019), reports a ranked probability score of 0.187. The original article does not report any standard deviations. Our findings are just a little higher $0.190(0.135)$, but again within the range of the standard deviations. For the random forest model, by Groll et al. (2018), a ranked probability score of 0.192 was reported. This within the magnitude of our findings of $0.195(0.136)$.

Groll et al. (2018, 2019) used multiple extra covariates, to which we had no access to and they had better data quality for them. This could explain the slightly higher ranked probability scores found in this master dissertation. For example, the FIFA

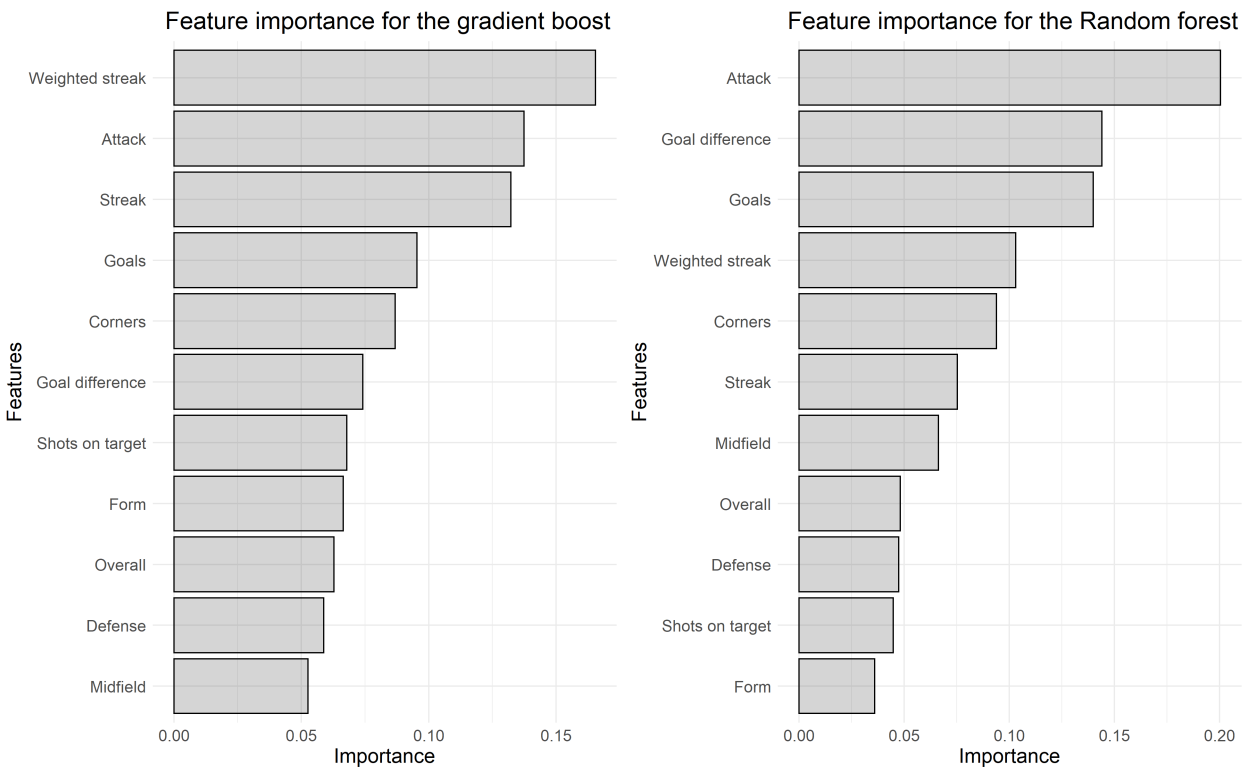

Figure 3.1: Feature importance of the gradient boost from Baboota and Kaur (2019)

ratings, obtained from Baboota and Kaur (2019), and the age of the players, which were obtained from *https://www.worldfootball.net*, proved challenging to merge with the datasets that contained other the covariates and the match outcomes. This might explain there lower feature importance of these features in our findings, compared to the original article (see figure 3.2).

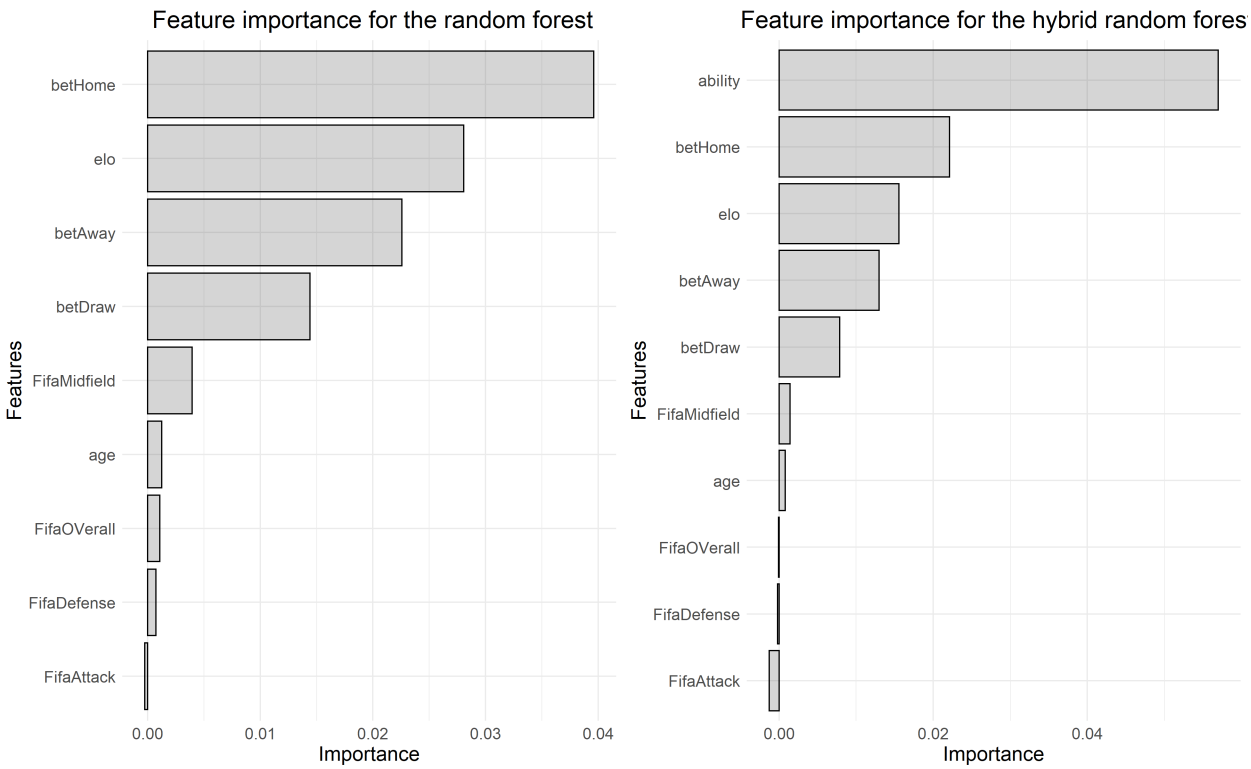

Figure 3.2: Feature importance of the random forest and hybrid random forest from Groll et al. (2018, 2019)

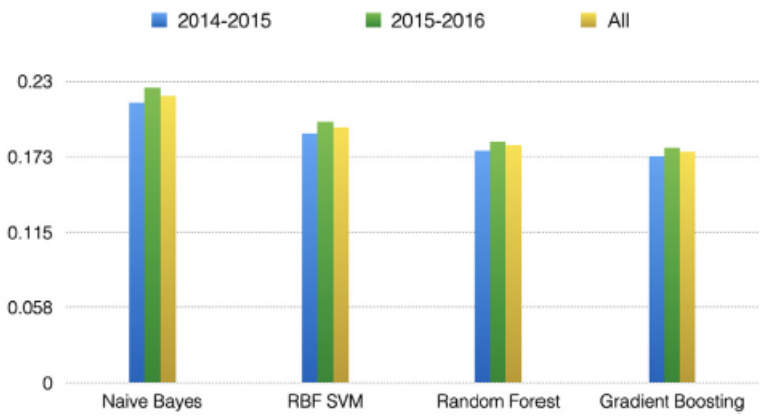

Figure 3.3: Ranked probability scores reported by Baboota and Kaur (2019)

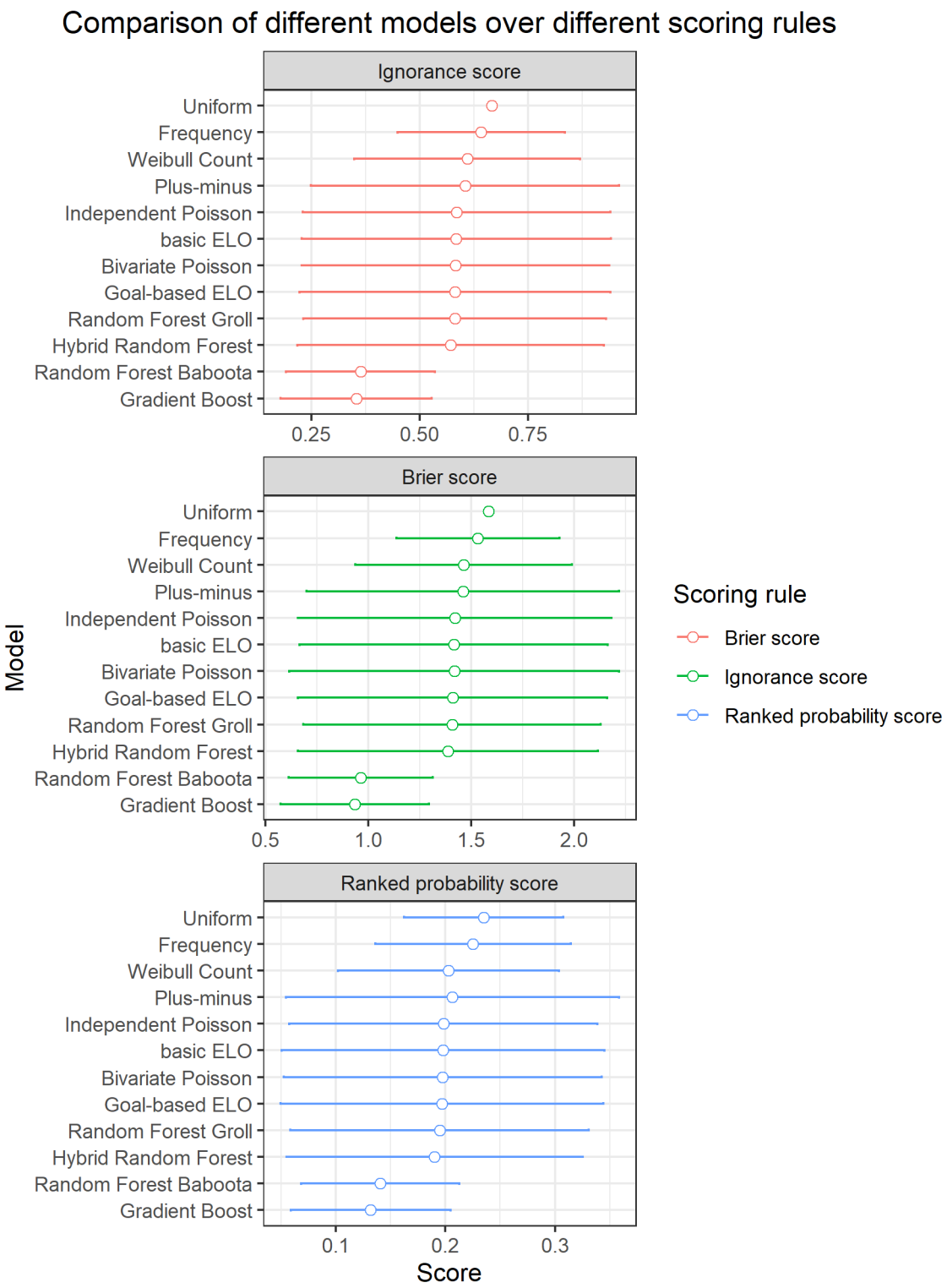

Figure 3.4: Comparison of the models over different scoring rules

CHAPTER 4

DISCUSSION

This chapter discusses the results found in this master dissertation. First, the controversial findings are discussed on a per model basis. After that, we give practical implications about the models and insights for future research.

4.1 ELO based models

For the ELO based models, our findings exceeded the expectations. We report better scores for both the Brier and ignorance score, compared to the original article of Hvattum and Arntzen (2010). As mentioned in 3.2, this might be due to the initial burn-in of five weeks. This burn-in procedure ensures that the models get some information on the new teams entering the league for that season. It is reasonable to assume that the estimations for the relative strength of the teams stabilise throughout a season. The predictions should thus increase in precision in the course of a season. Figure 4.1 shows the linear trends of the average of the scoring rule per week of a season. On this figure, the expected decrease is only observed for the ELO based model, but not for the models by Baboota and Kaur (2019).

Figure 4.1: Evolution of the scoring rules during the season of 2012

Another possible reason for our improved findings is the scale on which the ELO ratings are calculated. Hvattum and Arntzen (2010) calculated the ELO ratings over all the domestic leagues from England, namely the English Premier League, the Championship, League One and League Two. We specifically calculated only the ratings for the English Premier League on a per season basis, as described in section 2.3.2.

The ELO based models were very similar in performance compared to the Poisson based models. The goal-based ELO mildly outperforms the bivariate Poisson, and the basic ELO mildly outperforms the basic ELO. One possible reason for this might be the fact that the ELO based models indirectly got an additional season as data since we extended the initial protocol with one season to get initial reliable ELO ratings for the competing teams.

Although comparative studies are relatively absent in the current literature, Egidi and Torelli (2020) recently published an article where goal-based and result-based approaches where compared. The article compared two multinomial regression approaches to an independent and bivariate Poisson model, which were based on the models from Karlis and Ntzoufras (2003). Egidi and Torelli (2020) report that the result-based approaches, namely the two multinomial regressions, had higher Bries scores compared to the Poisson based alternatives. However, when predicting the World Cup tournament, Egidi and Torelli (2020) showed that the multinomial models outperformed the Poisson based models, by using the LOOIC (Leave-one-out cross-validation). The LOOIC is an approach for estimating the pointwise out-of-sample prediction accuracy from a fitted Bayesian model, by using the log-likelihood evaluated at the posterior simulations of the parameter value (Vehtari et al., 2017).

Egidi and Torelli (2020) point out that the multinomial models are less complex and that the outcome classes are easier to predict compared to the exact number of goals scored by each team. On the contrary, Ley et al. (2019) argued that goal-based models have a better performance since they incorporate additional information as the goal difference.

4.2 Plus minus ratings

The plus-minus ratings are a top-down rating system, that aims to divide the success of a team's performance onto the players responsible for it, which results in ratings for the individual players. To put this into perspective, for the prediction of the season 2015 a total of 1415 unique players were observed over 2889 matches, which are further divided into 17820 segments. For every segment, there are only 23 non-zero

data points over the features. Datasets with such high dimensions often have to deal with a problem called the curse of dimensionality (Bellman, 2003).

This curse of dimensionality occurs when the number of predictors increases, because then the distance between the data points increases exponentially. A regularised approach is thus a necessity for the plus-minus ratings. A potential downside of these regularised approaches is that they trade the reduction in variance for an increase in bias, for the parameter estimations. The high amount of parameters to estimate, in combination with the introduced bias, might explain why the model based on the plus-minus ratings is underperforming compared to the other models. Player ratings may thus be noisier compared to direct team ratings and not enhance predictions.

In a more recent study, Arntzen and Hvattum (2020) used an improved version of the plus-minus ratings, described in Pantuso and Hvattum (2019), as a covariate in an ordered logit regression. The improved plus-minus rating system also includes the effect of red cards and the age of the players. Interestingly, it also takes the differences in competitive leagues and player similarities into account, which allows for parameter estimation on a much broader scale. The plus-minus ratings might also become more reliable with an increase in data.

Arntzen and Hvattum (2020) also report that when both the ELO and plus-minus rating are combined in an ordered logit regression, both the Brier and ignorance score are significantly reduced, and thus the predictive performance is enhanced.

Finally, figure 4.2 shows that the time depreciation effect used by Arntzen and Hvattum (2020) pays more attention to older observations compared to the models from Ley et al. (2019) and Boshnakov et al. (2017). These higher weights for older observations might not be optimal for the small scale comparative study of this master dissertation.

4.3 Weibull count

Boshnakov et al. (2017) found that the Weibull count model had a better fit to the observed goal distribution compared to the Poisson based alternatives, according to the AIC criterion. However, this improved fit does not seem to be reflected in its predictive performance. A possible explanation for this could be that more parsimonious models are favoured over more complex ones. Ley et al. (2019) found that the independent and bivariate Poisson with one strength parameter outperformed their counterparts with two strength parameters, by using the RPS scoring rule.

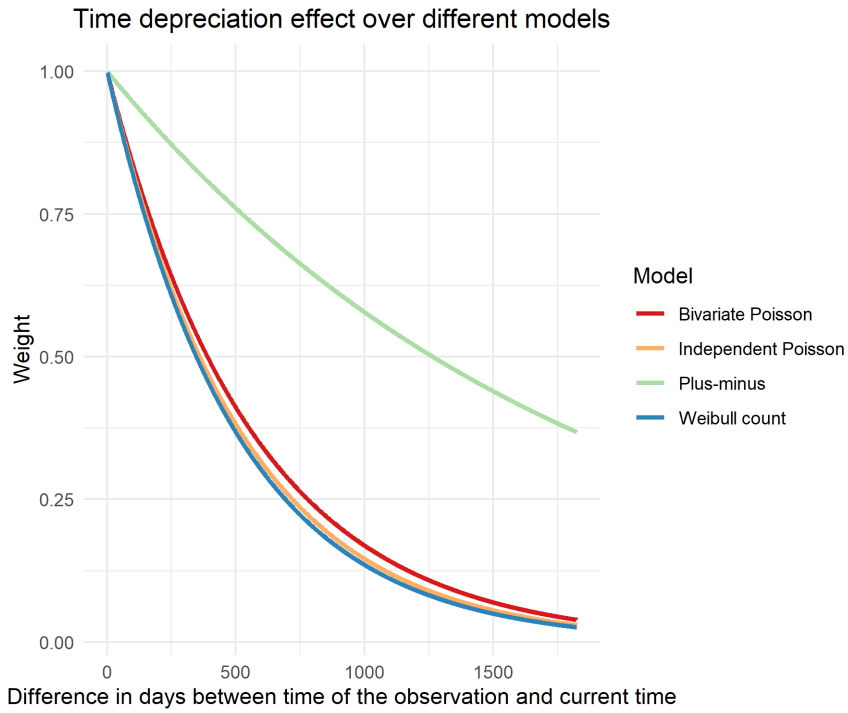

Figure 4.2: Time delay function for different models

Another possible reason for the lower performance of the Weibull count model is that it aims to estimate the parameters to have an optimal fit to the observed goal distributions, while the Poisson based models used by Ley et al. (2019) were mainly used to come up with an effective system to rank the opposing teams relative to each other. Ley et al. (2019) used some constraints in order to achieve this, e.g. the strengths of all the opposing teams were set to sum to zero. Boshnakov et al. (2017) did not use any constraints on the parameter estimations, which might make them unreliable and prone to learn noise.

4.4 Machine learning models

All the included models in this master dissertation that used more sophisticated machine learning techniques obtained better predictions. In particular, the models from Baboota and Kaur (2019) vastly outperformed the other models. The features for the models by Baboota and Kaur (2019) had better quality compared to the features used in the models by Groll et al. (2019), because for the features in the model by Groll et al. (2019) multiple datasets had to be merged by a unique key, which proved to be difficult. The features used in the models by Baboota and Kaur (2019) are published on the author's personal GitHub and were of high quality.

While most models are solely based on historical data, e.g. for the Poisson based models two years of data is needed to estimate the current abilities of the opposing teams, the features of Baboota and Kaur (2019) incorporate a mixture of features based on historical data (FIFA ranks) and features that capture the recent performance of a team. The importance of the features that capture the recent performances of teams might be exciting to investigate further since the scoring rules stayed relatively stable over the weeks during a season, see figure 4.1, and the variation is much smaller compared to the other models. The feature importance plots (see figure 3.1) also show that many of the features that capture the recent performance are essential for the accuracy of the model.

Interestingly, Baboota and Kaur (2019) give the suggestion to explore the gradient boosting and random forest regression approaches, which aim to estimate the goal distributions of each team. Groll et al. (2018) compared the random forest goal-based model, which predicts the intensity parameter λ for both opposing teams, to a result-based random forest model, which directly predicts the outcome classes, and found that the goal-based random forest models performed better than the result-based alternatives.

4.5 Hybrid models

The feature importance plots on figure 3.2 show the importance of a key engineered feature, namely the abilities of the teams. This ability parameter significantly increased the accuracy of the model when included, which shows the importance of incorporating highly informative features.

Both comparative studies and studies that aim at combining different models are relatively absent from the current literature. The hybrid random forest illustrates that rather than focusing on unique techniques or new engineered features to enhance predictions, scholars should focus on ensembled models.

4.6 Practical implications

We consider both the computing time and predictive performance to be valuable when considering the practical implications of the models. Football match prediction models are of great interest to decision-makers in the sport, football fans, news reporters and experts in the field.

Although single predictions from all models did not take longer than 45 seconds, when multiple matches or tournaments have to be predicted, the computation times can increase rapidly for specific models. The lowest computational time was obtained with the ELO-based models. The ELO-based models are built upon relatively simple calculations, which caused predictions of a season to take only a couple of seconds. Their standard results and swift computational times make them ideal for running quick simulations. The ELO parameters also proved to be of importance when used as a covariate in the random forest models from Groll et al. (2018, 2019) (see figure 3.2).

The computational times for the estimation of the ability parameters took around 20 seconds, depending on the amount of historical data incorporated. When making multiple predictions, these computational times can increase drastically and take up several hours. When choosing between the Poisson and ELO-based models, the ELO-based models are favoured, since they have equal performance with faster computations. However, the ability parameter incorporated in the hybrid random forest model proved to be the most important covariate. The advantage of the Poisson based models lies primarily in the creation of this highly informative ranking feature.

We do not recommend using the Weibull count model for predictions, since the computations were demanding and the performance was unsatisfactory. The attack and defence parameters, estimated as a result of the Weibull count model, could be further investigated in a random forest approach, to evaluate if they hold some value.

The random forest and boosting approaches had the best performances. The computational times were mostly dependent on the number of independent trees that had to be grown. The models of Baboota and Kaur (2019) only used 100 independent trees, while Groll et al. (2019) used 5000. Consequently, the individual predictions by the models of Baboota and Kaur (2019) took less than a second to complete, while for the models used by Groll et al. (2019) took around 30 seconds.

The plus-minus ratings were very promising, but due to a lack of data availability, the estimates were unreliable. It seems that for the prediction of football match outcomes, the player-based ratings can not yet compete with the team based ratings, and perhaps they are more useful when trying to evaluate the market value of players, as done in Sæbø and Hvattum (2015). Arntzen and Hvattum (2020) found that the combination of both the ELO ratings and plus-minus ratings significantly improved the predictive performance of the models, so it might be worth further investigating the plus-minus ratings in ensembled techniques.

To conclude, we give table 4.1, which ranks the models based on the performance and computational time. As mentioned above the more advanced machine learning

methods outperform the other models based on predictive power. The ELO-based models have an equal predictive performance as the Poisson-based models but obtain the results with much lower computational demands. The Weibull count and plus-minus rating model have the lowest predictive performance. From those two, we gave the lowest rank to the plus-minus ratings, since it requires much more computational time. Scholars and researchers in this field mainly try to develop new sophisticated features to rank the opposing teams or come up with advanced algorithms to predict football match results. We want to emphasise on the importance of combining various models and engineered features from the existing literature, which proves, in this master dissertation, to enhance the predictive power for most models.

Model	Performance	Computational time	Rank
Gradient boost	Very good	Medium	1
Random forest$_1$	Very good	Medium	2
Hybrid random forest	Good	Medium	3
Random forest$_2$	Good	Medium	4
Goal-based ELO	Average	Fast	5
Basic ELO	Average	Fast	6
Bivariate Poisson	Average	Medium	7
Independent Poisson	Average	Medium	8
Weibull count	Bad	Slow	9
Plus-minus	Bad	Very slow	10

Table 4.1: The final ranking of the models.
Random forest$_1$ from Baboota and Kaur (2019) and Random forest$_2$ from Groll et al. (2018)

BIBLIOGRAPHY

Arntzen, H. and Hvattum, L. M. (2020). Predicting match outcomes in association football using team ratings and player ratings. *Statistical Modelling*, page 1471082X20929881.

Baboota, R. and Kaur, H. (2019). Predictive analysis and modelling football results using machine learning approach for english premier league. *International Journal of Forecasting*, 35(2):741–755.

Baker, R., Boshnakov, G., Kharrat, T., and McHale, I. (2016). Countr: an r package to generate flexible count models. *Journal of Statistical Software*.

Bellman, R. (2003). Dynamic programming, 1957. *A very comprehensive reference with many economic examples is*.

Boshnakov, G., Kharrat, T., and McHale, I. G. (2017). A bivariate weibull count model for forecasting association football scores. *International Journal of Forecasting*, 33(2):458–466.

Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3.

Bröcker, J. and Smith, L. A. (2007). Scoring probabilistic forecasts: The importance of being proper. *Weather and Forecasting*, 22(2):382–388.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Constantinou, A. C. and Fenton, N. E. (2012). Solving the problem of inadequate scoring rules for assessing probabilistic football forecast models. *Journal of Quantitative Analysis in Sports*, 8(1).

Constantinou, A. C. and Fenton, N. E. (2013). Determining the level of ability of football teams by dynamic ratings based on the relative discrepancies in scores between adversaries. *Journal of Quantitative Analysis in Sports*, 9(1):37–50.

Constantinou, A. C., Fenton, N. E., and Neil, M. (2012). pi-football: A bayesian network model for forecasting association football match outcomes. *Knowledge-Based Systems*, 36:322–339.

Curley, J. P. (2016). engsoccerdata. English Soccer Data 1871-2016. R package version 0.1.5.

Dixon, M. J. and Coles, S. G. (1997). Modelling association football scores and inefficiencies in the football betting market. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 46(2):265–280.

Egidi, L. and Torelli, N. (2020). Comparing goal-based and result-based approaches in modelling football outcomes. *Social Indicators Research*, pages 1–13.

Epstein, E. S. (1969). A scoring system for probability forecasts of ranked categories. *Journal of Applied Meteorology*, 8(6):985–987.

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378.

Goddard, J. (2005). Regression models for forecasting goals and match results in association football. *International Journal of forecasting*, 21(2):331–340.

Groll, A., Ley, C., Schauberger, G., and Van Eetvelde, H. (2018). Prediction of the fifa world cup 2018-a random forest approach with an emphasis on estimated team ability parameters. *arXiv preprint arXiv:1806.03208*.

Groll, A., Ley, C., Schauberger, G., and Van Eetvelde, H. (2019). A hybrid random forest to predict soccer matches in international tournaments. *Journal of Quantitative Analysis in Sports*, 15(4):271–287.

Hvattum, L. M. (2017). Ordinal versus nominal regression models and the problem of correctly predicting draws in soccer. *International Journal of Computer Science in Sport*, 16(1):50–64.

Hvattum, L. M. (2019). A comprehensive review of plus-minus ratings for evaluating individual players in team sports. *International Journal of Computer Science in Sport*, 18(1):1–23.

Hvattum, L. M. and Arntzen, H. (2010). Using elo ratings for match result prediction in association football. *International Journal of forecasting*, 26(3):460–470.

Joseph, A., Fenton, N. E., and Neil, M. (2006). Predicting football results using bayesian nets and other machine learning techniques. *Knowledge-Based Systems*, 19(7):544–553.

Karlis, D. and Ntzoufras, I. (2003). Analysis of sports data by using bivariate poisson models. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(3):381–393.

Katti, S. and Rao, A. V. (1968). Handbook of the poisson distribution.

Keogh, F. and Rose, G. (2013). Football betting-the global gambling industry worth billions. *BBC, 3rd October, available at: http://www. bbc. com/sport/football/24354124 (accessed 6th July, 2016)*.

Koning, R. H. (2000). Balance in competition in dutch soccer. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 49(3):419–431.

Kundu, T., Choudhury, A. R., and Rai, S. (2019). Predicting english premier league matches using classification and regression.

Ley, C., Wiele, T. V. d., and Eetvelde, H. V. (2019). Ranking soccer teams on the basis of their current strength: A comparison of maximum likelihood approaches. *Statistical Modelling*, 19(1):55–73.

Macdonald, B. (2012). Adjusted plus-minus for nhl players using ridge regression with goals, shots, fenwick, and corsi. *Journal of Quantitative Analysis in Sports*, 8(3).

Maher, M. J. (1982). Modelling association football scores. *Statistica Neerlandica*, 36(3):109–118.

McHale, I. and Scarf, P. (2011). Modelling the dependence of goals scored by opposing teams in international soccer matches. *Statistical Modelling*, 11(3):219–236.

Milton, J. (2017). History of sports betting.

Mosteller, F. (2006). Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. In *Selected Papers of Frederick Mosteller*, pages 157–162. Springer.

Munting, R. (1996). *An economic and social history of gambling in Britain and the USA*. Manchester University Press.

Pantuso, G. and Hvattum, L. M. (2019). Maximizing performance with an eye on the finances: a chance-constrained model for football transfer market decisions. *arXiv preprint arXiv:1911.04689*.

Probst, P. and Boulesteix, A.-L. (2017). To tune or not to tune the number of trees in random forest. *The Journal of Machine Learning Research*, 18(1):6673–6690.

RightCasino (2014). History of online casinos: infographic.

Sæbø, O. D. and Hvattum, L. M. (2015). Evaluating the efficiency of the association football transfer market using regression based player ratings. In *Norsk IKT-konferanse for forskning og utdanning*.

Skellam, J. G. (1946). The frequency distribution of the difference between two poisson variates belonging to different populations. *Journal of the Royal Statistical Society. Series A (General)*, 109(Pt 3):296–296.

Stefani, R. T. (1977). Football and basketball predictions using least squares. *IEEE Transactions on systems, man, and cybernetics*, 7(2):117–21.

Suzuki, A. K., Salasar, L. E., Leite, J., and Louzada-Neto, F. (2010). A bayesian approach for predicting match outcomes: the 2006 (association) football world cup. *Journal of the Operational Research Society*, 61(10):1530–1539.

Thurstone, L. L. (1927). Psychophysical analysis. *The American journal of psychology*, 38(3):368–389.

Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and computing*, 27(5):1413–1432.

Wheatcroft, E. (2019). Evaluating probabilistic forecasts of football matches: The case against the ranked probability score. *arXiv preprint arXiv:1908.08980*.

APPENDIX A

APPENDIX

[Appendix]

A.1 Code for the scoring rules

```
BrierScore <- function(score){

  pwin <- score[1]
  pdraw <- score[2]
  ploss <- score[3]

  owin <- score[4]
  odraw <- score[5]
  oloss <- score[6]

  brier<- sum((pwin-owin)**2,
              (pdraw-odraw)**2,
              (ploss-oloss)**2)
  return(brier)
}


RPS2 <- function(score){

  pwin <- score[1]
  pdraw <- score[2]
  ploss <- score[3]

  owin <- score[4]
  odraw <- score[5]
  oloss <- score[6]

  rps2<- sum((pwin-owin)**2,
              (ploss-oloss)**2)
  return(rps2)
}

IGN <- function(score){
```

```

  pwin <- score[1]
  pdraw <- score[2]
  ploss <- score[3]

  owin <- score[4]
  odraw <- score[5]
  oloss <- score[6]

  if (owin == 1) {
    ignorance <- -log(base = 2,x = pwin)
  } else
    if (odraw == 1) {
      ignorance <- -log(base = 2,x = pdraw)
    } else ignorance <- -log(base = 2,x = ploss)

  return(ignorance)
}

IGN2 <- function(score){

  pwin <- score[1]
  pdraw <- score[2]
  ploss <- score[3]

  owin <- score[4]
  odraw <- score[5]
  oloss <- score[6]

  if (owin == 1) {
    ignorance <- -log(base = 2,x = pwin)
  } else
    if (odraw == 1) {
      ignorance <- -log(x = pdraw)
    } else ignorance <- -log(base = 2,x = ploss)

  return(ignorance)
}




RPS <- function(score, outcomes = 3){

  pwin <- score[1]
  pdraw <- score[2]
  ploss <- score[3]
```

```

  p <- c(pwin,pdraw,ploss)
  p <- unlist(p)
  p <- unname(p)

  owin <- score[4]
  odraw <- score[5]
  oloss <- score[6]

  o <- c(owin,odraw,oloss)
  o <- unlist(o)
  o <- unname(o)

  cumulative <- c()
  for (i in 1:outcomes-1){
    for (j in 1:i){
      cumulative <- c(cumulative , (p[j]-o[j])^2)
    }

  }

  return(sum(cumulative))
}
```

A.2 Code for the ELO based models

A.2.1 basic ELO

```
##############################################################
# Using ELO ratings for match result prediction in association football
#  M. hvattum
##############################################################

# packages
{
  setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/'
      )
  library(engsoccerdata)
  library(tidyverse)
  library(skellam)
  library(caret)
  library(MASS)
  source('BrierRpsIgnorance.R')
}

#data
{
```

```
  # Premier League
  PL_data <- england

  PL_data%>% filter(tier == 1)


  data <- PL_data %>%
    filter(tier == 1,
           Season %in% 2005:2017)

}

# Save output
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed_home <- c()
observed_draw <- c()
observed_away <- c()

# algorithm
for (season_id in 2008:2015){

  data %>% group_by(Season) %>%
    summarise(matches = length(Date),
              team = length(levels(factor(home))))


  #make round parameter
  id <- order(data$Season, data$Date)
  data <- data[id,]
  data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )



  # 1. initial elo ranking
  {
    Init_data <- data %>% filter(Season == season_id-3)


    teams <- sort(levels(factor(Init_data$home)))
    elo <- rep(0, length(teams))

    elo_hash <- data.frame(team = teams, elo = elo)

    for (i in 1:nrow(Init_data)){

      # 1 . Hyperparameters
      c <- 10
```

```
    d <- 400
    k <- 20

    # 2. observation
    obs <- Init_data[i,]

    elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
    elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]
    x <- elo_home - elo_away

    E_score_home <- (1+c^(-x/d))^-1
    E_score_away <- 1 - E_score_home

    actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                ifelse(obs$result == 'A', 0, 0.5 ))
    actual_score_away <- 1 - actual_score_home

    # 3. Update Elo
    elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
    elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

    elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_update
    elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_update

  }

}


# 2. Training data set
{
  train_data <-  data %>%
    filter(Season %in% (season_id-2):(season_id-1))

  train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))

  print('training dataset')
  print(train_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))


  teams <- levels(factor(train_data$home))[!levels(factor(train_data$home)) %in% elo_hash$team]

  elo_hash <- rbind(elo_hash,
        data.frame(team = teams,
```

```
                elo = rep(0,length(teams))))


    x_vect <- c()
    y_vect <- c()

    for (i in 1:nrow(train_data)){

      # 1. Hyperparameters
      c <- 10
      d <- 400
      k <- 20

      # 2. Draw observation and save information
      obs <- train_data[i,]

      y_vect <- c(y_vect,as.character(obs$result))

      elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
      elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]

      x <- elo_home - elo_away
      x_vect <- c(x_vect,x)

      E_score_home <- (1+c^(-x/d))^-1
      E_score_away <- 1 - E_score_home

      actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                  ifelse(obs$result == 'A', 0, 0.5 ))
      actual_score_away <- 1 - actual_score_home

      # 3. Update Elo
      elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
      elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

      elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_
          update
      elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_
          update

    }

    # Use information for OLR
    OLR <- polr(formula = 'y~x',
                data = data.frame(y = factor(y_vect, levels = c('A','D','H'),ordered
                     = TRUE),
                                  x = x_vect))

    summary(OLR)
```

```
}


# 3. Test dataset
{
  test_data <- data %>% filter(Season == season_id,round > 5)
  print(test_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor
      (round)))))

  for (i in 1:nrow(test_data)){

    # 1. Hyper parameters
    c <- 10
    d <- 400
    k <- 20

    # 2. Match to predict
    obs <- test_data[i,]

    y_vect <- c(y_vect,as.character(obs$result))

    elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
    elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]

    x <- elo_home - elo_away
    x_vect <- c(x_vect,x)

    # 3. Predict
    prob <- predict(OLR,
                    newdata = data.frame(y = factor(obs$result, levels = c('A','D'
                        ,'H'),ordered = TRUE),
                                         x = x),
                    type = 'p')

    Prob_home <- c(Prob_home,unname(prob['H']))
    Prob_draw <- c(Prob_draw,unname(prob['D']))
    Prob_away <- c(Prob_away,unname(prob['A']))

    # 4. observed
    if (as.character(obs$result) == 'H'){
      observed_home <- c(observed_home,1)
      observed_draw <- c(observed_draw,0)
      observed_away <- c(observed_away,0)

    }
    else {
      if (as.character(obs$result) == 'A'){
        observed_home <- c(observed_home,0)
        observed_draw <- c(observed_draw,0)
```

```
          observed_away <- c(observed_away,1)

        }
        else {
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,1)
          observed_away <- c(observed_away,0)

        }
      }

      # 5. Update
      E_score_home <- (1+c^(-x/d))^-1
      E_score_away <- 1 - E_score_home

      actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                  ifelse(obs$result == 'A', 0, 0.5 ))
      actual_score_away <- 1 - actual_score_home

      elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
      elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

      elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_
          update
      elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_
          update

      OLR <- polr(formula = 'y~x',
                  data = data.frame(y = factor(y_vect, levels = c('A','D','H'),
                      ordered = TRUE),
                                    x = x_vect))
      }

    }
}

out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away)

out$ignorance <- apply(out[,1:6], 1, IGN)
out$brierScore <-apply(out[,1:6], 1, BrierScore)
out$RPS <- apply(out[,1:6], 1, RPS2)


mean(out$ignorance)
```

```
sd(out$ignorance)

mean(out$brierScore)
sd(out$brierScore)

mean(out$RPS/2)
sd(out$RPS/2)

saveRDS(object = out,file = 'ELOb_Hvattum2010.rds')
```

A.2.2 Goal-based ELO

```
##########################################################
# Using ELO ratings for match result prediction in association football
#  M. hvattum
##########################################################

# packages
setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/')
library(engsoccerdata)
library(tidyverse)
library(skellam)
library(caret)
library(MASS)
source('BrierRpsIgnorance.R')


# Premier League
PL_data <- england

PL_data%>% filter(tier == 1)


data <- PL_data %>%
  filter(tier == 1,
         Season %in% 2005:2017)



# Save output
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed_home <- c()
observed_draw <- c()
observed_away <- c()
week <- c()

# algorithm
```

```
for (season_id in 2008:2015){

  data %>% group_by(Season) %>%
    summarise(matches = length(Date),
              team = length(levels(factor(home))))


  #make round parameter
  id <- order(data$Season, data$Date)
  data <- data[id,]
  data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )



  # 1. initial elo ranking
  {
    Init_data <- data %>% filter(Season == season_id-3)


    teams <- sort(levels(factor(Init_data$home)))
    elo <- rep(0, length(teams))

    elo_hash <- data.frame(team = teams, elo = elo)

    for (i in 1:nrow(Init_data)){

      # 1. observation
      obs <- Init_data[i,]

      # 2 . Hyperparameters
      c <- 10
      d <- 400
      k0 <- 10
      lambda <- 1
      gamma <- abs(obs$hgoal-obs$vgoal)
      k <- k0*((1+gamma)**lambda)

      # 3. variables
      elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
      elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]
      x <- elo_home - elo_away

      E_score_home <- (1+c^(-x/d))^-1
      E_score_away <- 1 - E_score_home

      actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                  ifelse(obs$result == 'A', 0, 0.5 ))
      actual_score_away <- 1 - actual_score_home

```

```
      # 4. Update Elo
      elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
      elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

      elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_update
      elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_update

    }

  }


  # 2. Training data set
  {
    train_data <-  data %>%
      filter(Season %in% (season_id-2):(season_id-1))

    train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))

    print('training dataset')
    print(train_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))


    teams <- levels(factor(train_data$home))[!levels(factor(train_data$home)) %in% elo_hash$team]

    elo_hash <- rbind(elo_hash,
                      data.frame(team = teams,
                                 elo = rep(0,length(teams))))


    x_vect <- c()
    y_vect <- c()

    for (i in 1:nrow(train_data)){

      # 1. Draw observation and save information
      obs <- train_data[i,]

      y_vect <- c(y_vect,as.character(obs$result))

      elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
      elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]

      x <- elo_home - elo_away
```

```
      x_vect <- c(x_vect,x)

      E_score_home <- (1+c^(-x/d))^-1
      E_score_away <- 1 - E_score_home

      actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                  ifelse(obs$result == 'A', 0, 0.5 ))
      actual_score_away <- 1 - actual_score_home


      # 2. Hyperparameters
      c <- 10
      d <- 400
      k0 <- 10
      lambda <- 1
      gamma <- abs(obs$hgoal-obs$vgoal)
      k <- k0*((1+gamma)**lambda)


      # 3. Update Elo
      elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
      elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

      elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_
          update
      elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_
          update

    }

    # Use information for OLR
    OLR <- polr(formula = 'y~x',
                data = data.frame(y = factor(y_vect, levels = c('A','D','H'),ordered
                     = TRUE),
                                  x = x_vect))

    summary(OLR)
  }


  # 3. Test dataset
  {
    test_data <- data %>% filter(Season == season_id,round > 5)
    print(test_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor
        (round)))))

    for (i in 1:nrow(test_data)){

      week <- c(week,test_data$round)
```

```r

    # 1. Match to predict
    obs <- test_data[i,]

    y_vect <- c(y_vect,as.character(obs$result))

    elo_home <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)]
    elo_away <- elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)]

    x <- elo_home - elo_away
    x_vect <- c(x_vect,x)

    E_score_home <- (1+c^(-x/d))^-1
    E_score_away <- 1 - E_score_home

    actual_score_home <- ifelse(obs$result == 'H' , 1 ,
                                ifelse(obs$result == 'A', 0, 0.5 ))
    actual_score_away <- 1 - actual_score_home



    # 2. Hyper parameters
    c <- 10
    d <- 400
    k0 <- 10
    lambda <- 1
    gamma <- abs(obs$hgoal-obs$vgoal)
    k <- k0*((1+gamma)**lambda)


    # 3. Predict
    prob <- predict(OLR,
                    newdata = data.frame(y = factor(obs$result, levels = c('A','D'
                        ,'H'),ordered = TRUE),
                                         x = x),
                    type = 'p')

    Prob_home <- c(Prob_home,unname(prob['H']))
    Prob_draw <- c(Prob_draw,unname(prob['D']))
    Prob_away <- c(Prob_away,unname(prob['A']))

    # 4. observed
    if (as.character(obs$result) == 'H'){
      observed_home <- c(observed_home,1)
      observed_draw <- c(observed_draw,0)
      observed_away <- c(observed_away,0)

    }
    else {
```

```
        if (as.character(obs$result) == 'A'){
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,0)
          observed_away <- c(observed_away,1)

        }
        else {
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,1)
          observed_away <- c(observed_away,0)

        }
      }

      # 5. Update

      elo_home_update <- elo_home + k*(actual_score_home-E_score_home)
      elo_away_update <- elo_away + k*(actual_score_away-E_score_away)

      elo_hash$elo[levels(elo_hash$team) == as.character(obs$home)] <- elo_home_
          update
      elo_hash$elo[levels(elo_hash$team) == as.character(obs$visitor)] <- elo_away_
          update

      OLR <- polr(formula = 'y~x',
                  data = data.frame(y = factor(y_vect, levels = c('A','D','H'),
                      ordered = TRUE),
                                    x = x_vect))
    }

  }
}

out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away)

out$ignorance <- apply(out[,1:6], 1, IGN)
out$brierScore <-apply(out[,1:6], 1, BrierScore)
out$RPS <- apply(out[,1:6], 1, RPS2)


mean(out$ignorance)
sd(out$ignorance)

mean(out$brierScore)
```

```
sd(out$brierScore)

mean(out$RPS/2)
sd(out$RPS/2)

saveRDS(object = out,file = 'ELOg_Hvattum2010.rds')



###################
#plot

out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away,
                  week = week)

out$IgnoranceScore <- apply(out[,1:6], 1, IGN)
out$BrierScore <-apply(out[,1:6], 1, BrierScore)
out$RankedProbScore <- apply(out[,1:6], 1, RPS2)


out$Model <- 'out'
out <- out[,c('IgnoranceScore','BrierScore','RankedProbScore', 'week')] %>% gather(
    key = 'ScoringRule' , value = 'Score',IgnoranceScore:RankedProbScore  )

p <- out %>%
  ggplot(aes(x= week, y=Score, color = ScoringRule, fill = ScoringRule)) +
  #geom_point(alpha = 0.25) +
  geom_smooth(method='glm',se = F)+
  theme_bw() +
  labs(x = 'Week',
       y ='Score',
       title = 'Evolution of the scoring rules over season 2012',
       color = 'Scoring rule',
       fill = 'Scoring rule') +
    theme(plot.title = element_text(hjust = 0.5))
p
ggsave(plot = p, 'scorerulee.png',height = 4,width = 5.2)
```

A.3 Code for the plus-minus models

A.3.1 Plus-minus

```
1 ##########################################################
2 # Player Ratings
3 # M. hvattum
4 ##########################################################
5
6 # packages
7 {
8   setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/'
        )
9   library(engsoccerdata)
10  library(tidyverse)
11  library(skellam)
12  library(caret)
13  library(MASS)
14  source('BrierRpsIgnorance.R')
15 }
16
17 # data
18 {
19
20  load('eplhvattum.Rdata')
21  data <- data1
22  str(data)
23  data <- data[data$date <= as.Date('2016-05-16'),]
24
25
26  # current times
27  # Premier League
28  PL_data <- england
29
30  PL_data <- PL_data%>% filter(tier == 1,
31                               Season %in% 2000:2017)
32
33  id <- order(PL_data$Season, PL_data$Date)
34  PL_data <- PL_data[id,]
35  PL_data$round <- rep(sort(rep(1:38,10)), max(PL_data$Season)- min(PL_data$Season)
        +1 )
36  DataSeasonRound <- unique(PL_data[,c('Date','Season','round')])
37  colnames(DataSeasonRound) <- c('date','season','round')
38  DataSeasonRound$date <- as.Date(DataSeasonRound$date)
39
40  data$Season <- DataSeasonRound$season[match(data$date,DataSeasonRound$date)]
41  data$round <- DataSeasonRound$round[match(data$date,DataSeasonRound$date)]
42
43 }
44
45 # functions
46 {
47  SplitSegments <- function(obs){
```

```r

  # information
  {
    # player changes
    {
      playerChanges <- data.frame(time = c(obs$home_change_minute1,obs$home_change_minute2,obs$home_change_minute3,
                                           obs$away_change_minute1,obs$away_change_minute2,obs$away_change_minute3),
                                  athleteIn = c(obs$home_change_in1,obs$home_change_in2,obs$home_change_in3,
                                                obs$away_change_in1,obs$away_change_in2,obs$away_change_minute3),
                                  athleteOut = c(obs$home_change_out1,obs$home_change_out2,obs$home_change_out3,
                                                 obs$away_change_out1,obs$away_change_out2,obs$away_change_out3))

      playerChanges <- playerChanges[order(playerChanges$time),]
      playerChanges <- na.omit(playerChanges)
    }

    # goal changes
    {

      Goalchanges <- data.frame(goals = c(obs$home_goal_minute1,obs$home_goal_minute2,obs$home_goal_minute3,
                                          obs$home_goal_minute4,obs$home_goal_minute5,obs$home_goal_minute6,
                                          obs$home_goal_minute7,obs$home_goal_minute8,obs$home_goal_minute9,
                                          obs$away_goal_minute1,obs$away_goal_minute2),
                                team = c(rep('home',9),rep('away',2)))
      Goalchanges <- Goalchanges[order(Goalchanges$goals),]
      Goalchanges <- na.omit(Goalchanges)
    }


    # params
    {
      segments <- list()
      start <- 0
      time <- 0
    }

    #players
```

```
    {
      players <- unique(c(levels(obs$home_team_player1),levels(obs$home_team_player2),levels(obs$home_team_player3),levels(obs$home_team_player4),
                          levels(obs$home_team_player5),levels(obs$home_team_player6),levels(obs$home_team_player7),levels(obs$home_team_player8),
                          levels(obs$home_team_player9),levels(obs$home_team_player10),levels(obs$home_team_player11)))

      matrix_players <- t(matrix(c(players, rep(0,length(players))),ncol =  2))

      home_players <- c(as.character(obs$home_team_player1),as.character(obs$home_team_player2),as.character(obs$home_team_player3),
                        as.character(obs$home_team_player4),as.character(obs$home_team_player5),as.character(obs$home_team_player6),
                        as.character(obs$home_team_player7),as.character(obs$home_team_player8),as.character(obs$home_team_player9),
                        as.character(obs$home_team_player10),as.character(obs$home_team_player11))
      away_players <- c(as.character(obs$away_team_player1),as.character(obs$away_team_player2),as.character(obs$away_team_player3),
                        as.character(obs$away_team_player4),as.character(obs$away_team_player5),as.character(obs$away_team_player6),
                        as.character(obs$away_team_player7),as.character(obs$away_team_player8),as.character(obs$away_team_player9),
                        as.character(obs$away_team_player10),as.character(obs$away_team_player11))

      matrix_players[2,match(home_players,matrix_players[1,])] <- 1
      matrix_players[2,match(away_players,matrix_players[1,])] <- -1

    }

  }


  if (nrow(playerChanges) > 0){
    for (i in 1:(nrow(playerChanges))){

      #segment duration
      end <- playerChanges$time[i]
      duration <- end - start
      start <- end
      if (duration == 0) duration <- 1

      #goals
      starttime <- time
      time <- time + duration
      endtime <- time
```

```
        goals_start <- sum(ifelse(Goalchanges$team[Goalchanges$goals<starttime] == 'home' , 1,-1))
        goals_end <- sum(ifelse(Goalchanges$team[Goalchanges$goals<endtime] == 'home' , 1,-1))
        gd <- goals_end - goals_start

        #beta
        beta <- 90*gd/duration

        #players
        a <- as.numeric(matrix_players[2,])


        # red cards


        # age


        #home
        homeeffect <- 1

        # add to segment list
        segment <- data.frame(matrix(c(beta,a,homeeffect,as.Date(obs$date),obs$Season,obs$round),nrow=1))
        colnames(segment) <-  c('beta',matrix_players[1,],'homeadv','date','Season','round')
        segments[[paste0('segment',i)]] <- segment

        # update players
        if (as.character(playerChanges$athleteOut[i]) %in% away_players){
          away_players[match(as.character(playerChanges$athleteOut[i]),away_players)] <- as.character(playerChanges$athleteIn[i])
        }
        if (as.character(playerChanges$athleteOut[i]) %in% home_players){
          home_players[match(as.character(playerChanges$athleteOut[i]),home_players)] <- as.character(playerChanges$athleteIn[i])
        }

        matrix_players <- t(matrix(c(players, rep(0,length(players))),ncol =  2))
        matrix_players[2,match(home_players,matrix_players[1,])] <- 1
        matrix_players[2,match(away_players,matrix_players[1,])] <- -1

      }
    }
    else { i <-1 }

    # last segment
    {
```

```
      #segment duration
      end <- 90
      duration <- end - start
      start <- end
      if (duration == 0) duration <- 1
      #goals
      starttime <- time
      time <- time + duration
      endtime <- time
      goals_start <- sum(ifelse(Goalchanges$team[Goalchanges$goals<starttime] == 'home' , 1,-1))
      goals_end <- sum(ifelse(Goalchanges$team[Goalchanges$goals<endtime] == 'home' , 1,-1))
      gd <- goals_end - goals_start

      #beta
      beta <- 90*gd/duration

      #players
      a <- as.numeric(matrix_players[2,])

      # red cards


      # age


      #home
      homeeffect <- 1

      segment <- data.frame(matrix(c(beta,a,homeeffect,obs$date,obs$Season,obs$round),nrow=1))
      colnames(segment) <-  c('beta',matrix_players[1,],'homeadv','date','Season','round')
      segments[[paste0('segment',i+1)]] <- segment

    }

    return(segments)
  }
}

#split data in segments
{
df <- data
df$home_team_player1 <- factor(df$home_team_player1)
df$home_team_player2 <- factor(df$home_team_player2)
df$home_team_player3 <- factor(df$home_team_player3)
df$home_team_player4 <- factor(df$home_team_player4)
```

```
df$home_team_player5 <- factor(df$home_team_player5)
df$home_team_player6 <- factor(df$home_team_player6)
df$home_team_player7 <- factor(df$home_team_player7)
df$home_team_player8 <- factor(df$home_team_player8)
df$home_team_player9 <- factor(df$home_team_player9)
df$home_team_player10 <- factor(df$home_team_player10)
df$home_team_player11 <- factor(df$home_team_player11)

df$away_team_player1 <- factor(df$away_team_player1)
df$away_team_player2 <- factor(df$away_team_player2)
df$away_team_player3 <- factor(df$away_team_player3)
df$away_team_player4 <- factor(df$away_team_player4)
df$away_team_player5 <- factor(df$away_team_player5)
df$away_team_player6 <- factor(df$away_team_player6)
df$away_team_player7 <- factor(df$away_team_player7)
df$away_team_player8 <- factor(df$away_team_player8)
df$away_team_player9 <- factor(df$away_team_player9)
df$away_team_player10 <- factor(df$away_team_player10)
df$away_team_player11 <- factor(df$away_team_player11)
#str(df)

seglist <- list()
for (id in 1:nrow(df)){
  obs <- df[id,]
  segments <- SplitSegments(obs)
  segdf <- do.call(rbind, segments)
  seglist[[as.character(id)]] <- segdf
}

df <- do.call(rbind, seglist)
}


df_2015 <- df %>% filter(Season %in% 2008:2015)
data2015 <- data %>% filter(Season %in% 2008:2015)

# Save output
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed <- c()

# algorithm
yrsago <- 4
for (season_id in 2008:2015){

  # train data
  train_data <-  data %>% filter(Season %in% (season_id-2):(season_id-1))

```

```
train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))
train_data$seasonround <- paste0(train_data$Season,train_data$round)

print('training dataset')
print(train_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))

## get results and pm
train_results <- c()
train_pm <- c()
for (i in 1:length(unique(train_data$seasonround))){

  #train obs
  train_obs <-  train_data[train_data$seasonround==unique(train_data$seasonround)[i],]

  print(paste('train',train_obs$Season[1],i))
  currenttime <- max(train_obs$date)

  # df of all segments before current time
  {
    df_seg_train <- df %>% filter((date <= currenttime)&((date >= currenttime - yrsago*365)))
    k <- 0.2
    time_depre <- exp(-k*(as.numeric(currenttime) - df_seg_train$date)/365)
  }

  # get coefficients
  {
    beta <- df_seg_train$beta*time_depre

    a <- as.matrix(df_seg_train[,-c(match('beta',colnames(df_seg_train)),
                                    match('Season',colnames(df_seg_train)),
                                    match('date',colnames(df_seg_train)),
                                    match('round',colnames(df_seg_train)))])

    a <- a * time_depre


    lambda <- 3000
    x <- solve(t(a) %*% a + diag(lambda,ncol(a))) %*% t(a) %*% beta
  }

  # for every obs of in currenttime get result and difference in pm of teams
  {
    gd <- train_obs$home_goals - train_obs$away_goals
    result <- ifelse(gd < 0, 'A', ifelse(gd == 0, 'D', 'H'))
    train_results <- c(train_results,result)

```

```
x_home_player1 <- x[match(as.character(train_obs$home_team_player1),rownames(x))]
x_home_player2 <- x[match(as.character(train_obs$home_team_player2),rownames(x))]
x_home_player3 <- x[match(as.character(train_obs$home_team_player3),rownames(x))]
x_home_player4 <- x[match(as.character(train_obs$home_team_player4),rownames(x))]
x_home_player5 <- x[match(as.character(train_obs$home_team_player5),rownames(x))]
x_home_player6 <- x[match(as.character(train_obs$home_team_player6),rownames(x))]
x_home_player7 <- x[match(as.character(train_obs$home_team_player7),rownames(x))]
x_home_player8 <- x[match(as.character(train_obs$home_team_player8),rownames(x))]
x_home_player9 <- x[match(as.character(train_obs$home_team_player9),rownames(x))]
x_home_player10 <- x[match(as.character(train_obs$home_team_player10),rownames(x))]
x_home_player11 <- x[match(as.character(train_obs$home_team_player11),rownames(x))]
x_home_player12 <- x[match(rep('homeadv',nrow(train_obs)),rownames(x))]
x_home <- apply(matrix(c(x_home_player1,x_home_player2,x_home_player3,
                         x_home_player4,x_home_player5,x_home_player6,
                         x_home_player7,x_home_player8,x_home_player9,
                         x_home_player10,x_home_player11,x_home_player12),
                       ncol = 12),
                1, mean,na.rm = TRUE)
x_home[is.na(x_home)] <- 0

x_away_player1 <- x[match(as.character(train_obs$away_team_player1),rownames(x))]
x_away_player2 <- x[match(as.character(train_obs$away_team_player2),rownames(x))]
x_away_player3 <- x[match(as.character(train_obs$away_team_player3),rownames(x))]
x_away_player4 <- x[match(as.character(train_obs$away_team_player4),rownames(x))]
x_away_player5 <- x[match(as.character(train_obs$away_team_player5),rownames(x))]
x_away_player6 <- x[match(as.character(train_obs$away_team_player6),rownames(x))]
x_away_player7 <- x[match(as.character(train_obs$away_team_player7),rownames(x))]
x_away_player8 <- x[match(as.character(train_obs$away_team_player8),rownames(x))]
x_away_player9 <- x[match(as.character(train_obs$away_team_player9),rownames(x))]
```

```
    x_away_player10 <- x[match(as.character(train_obs$away_team_player10),rownames(x))]
    x_away_player11 <- x[match(as.character(train_obs$away_team_player11),rownames(x))]
    x_away <- apply(matrix(c(x_away_player1,x_away_player2,x_away_player3,
                             x_away_player4,x_away_player5,x_away_player6,
                             x_away_player7,x_away_player8,x_away_player9,
                             x_away_player10,x_away_player11),
                           ncol = 11),
                     1, mean,na.rm = TRUE)
    x_away[is.na(x_away)] <- 0

    pmdiff <- x_home - x_away
    train_pm <- c(train_pm,pmdiff)
  }

}

# Use information for OLR
OLR <- polr(formula = 'result~pmdiff',
            data = data.frame(result = factor(train_results, levels = c('A','D','H'),ordered = TRUE),
                              pmdiff = train_pm))


# test data
test_data <- data %>% filter(Season == season_id,round > 5)
print(test_data %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))

for(round_id in 6:max(test_data$round)){


  #obs
  obs_test <- test_data %>% filter(round == round_id)
  print(paste('test',obs_test$Season[1],round_id))

  # for every obs of in currenttime get result and prob
  {
    gd <- obs_test$home_goals - obs_test$away_goals
    result <- ifelse(gd < 0, 'A', ifelse(gd == 0, 'D', 'H'))
    #train_results <- c(train_results,result)

    x_home_player1 <- x[match(as.character(obs_test$home_team_player1),rownames(x))]
    x_home_player2 <- x[match(as.character(obs_test$home_team_player2),rownames(x))]
    x_home_player3 <- x[match(as.character(obs_test$home_team_player3),rownames(x))]
```

```
x_home_player4 <- x[match(as.character(obs_test$home_team_player4),rownames(x))]
x_home_player5 <- x[match(as.character(obs_test$home_team_player5),rownames(x))]
x_home_player6 <- x[match(as.character(obs_test$home_team_player6),rownames(x))]
x_home_player7 <- x[match(as.character(obs_test$home_team_player7),rownames(x))]
x_home_player8 <- x[match(as.character(obs_test$home_team_player8),rownames(x))]
x_home_player9 <- x[match(as.character(obs_test$home_team_player9),rownames(x))]
x_home_player10 <- x[match(as.character(obs_test$home_team_player10),rownames(x))]
x_home_player11 <- x[match(as.character(obs_test$home_team_player11),rownames(x))]
x_home_player12 <- x[match(rep('homeadv',nrow(obs_test)),rownames(x))]
x_home <- apply(matrix(c(x_home_player1,x_home_player2,x_home_player3,
                         x_home_player4,x_home_player5,x_home_player6,
                         x_home_player7,x_home_player8,x_home_player9,
                         x_home_player10,x_home_player11,x_home_player12),
                       ncol = 12),
                1, mean,na.rm = TRUE)
x_home[is.na(x_home)] <- 0

x_away_player1 <- x[match(as.character(obs_test$away_team_player1),rownames(x))]
x_away_player2 <- x[match(as.character(obs_test$away_team_player2),rownames(x))]
x_away_player3 <- x[match(as.character(obs_test$away_team_player3),rownames(x))]
x_away_player4 <- x[match(as.character(obs_test$away_team_player4),rownames(x))]
x_away_player5 <- x[match(as.character(obs_test$away_team_player5),rownames(x))]
x_away_player6 <- x[match(as.character(obs_test$away_team_player6),rownames(x))]
x_away_player7 <- x[match(as.character(obs_test$away_team_player7),rownames(x))]
x_away_player8 <- x[match(as.character(obs_test$away_team_player8),rownames(x))]
x_away_player9 <- x[match(as.character(obs_test$away_team_player9),rownames(x))]
x_away_player10 <- x[match(as.character(obs_test$away_team_player10),rownames(x))]
x_away_player11 <- x[match(as.character(obs_test$away_team_player11),rownames(x))]
x_away <- apply(matrix(c(x_away_player1,x_away_player2,x_away_player3,
                         x_away_player4,x_away_player5,x_away_player6,
```

```
                              x_away_player7,x_away_player8,x_away_player9,
                              x_away_player10,x_away_player11),
                            ncol = 11),
                     1, mean,na.rm = TRUE)

    pmdiff <- x_home - x_away
    prob <- predict(OLR,
                     newdata = data.frame(result = factor(result, levels = c('A','D
                         ','H'),ordered = TRUE),
                                          pmdiff = pmdiff),
                     type = 'p')

    Prob_home <- c(Prob_home, as.numeric(prob[,3]))
    Prob_draw <- c(Prob_draw, as.numeric(prob[,2]))
    Prob_away <- c(Prob_away, as.numeric(prob[,1]))
    observed <- c(observed,result)

  }

  # add test to train and update coefficients
  train_data$seasonround <- NULL
  train_data <- rbind(train_data,obs_test)

  currenttime <- max(train_data$date)

  {
    df_seg_train <- df %>% filter((date <= currenttime)&((date >= currenttime -
        yrsago*365)))
    k <- 0.2
    time_depre <- exp(-k*(as.numeric(currenttime) - df_seg_train$date)/365)
  }

  # get coefficients
  {
    beta <- df_seg_train$beta*time_depre

    a <- as.matrix(df_seg_train[,-c(match('beta',colnames(df_seg_train)),
                                    match('Season',colnames(df_seg_train)),
                                    match('date',colnames(df_seg_train)),
                                    match('round',colnames(df_seg_train)))])

    a <- a * time_depre


    lambda <- 3000
    x <- solve(t(a) %*% a + diag(lambda,ncol(a))) %*% t(a) %*% beta
  }

  # for every obs of in currenttime get result and difference in pm of teams
```

```
  {
    gd <- obs_test$home_goals - obs_test$away_goals
    result <- ifelse(gd < 0, 'A', ifelse(gd == 0, 'D', 'H'))
    train_results <- c(train_results,result)

    x_home_player1 <- x[match(as.character(obs_test$home_team_player1),rownames(x)
        )]
    x_home_player2 <- x[match(as.character(obs_test$home_team_player2),rownames(x)
        )]
    x_home_player3 <- x[match(as.character(obs_test$home_team_player3),rownames(x)
        )]
    x_home_player4 <- x[match(as.character(obs_test$home_team_player4),rownames(x)
        )]
    x_home_player5 <- x[match(as.character(obs_test$home_team_player5),rownames(x)
        )]
    x_home_player6 <- x[match(as.character(obs_test$home_team_player6),rownames(x)
        )]
    x_home_player7 <- x[match(as.character(obs_test$home_team_player7),rownames(x)
        )]
    x_home_player8 <- x[match(as.character(obs_test$home_team_player8),rownames(x)
        )]
    x_home_player9 <- x[match(as.character(obs_test$home_team_player9),rownames(x)
        )]
    x_home_player10 <- x[match(as.character(obs_test$home_team_player10),rownames(
        x))]
    x_home_player11 <- x[match(as.character(obs_test$home_team_player11),rownames(
        x))]
    x_home_player12 <- x[match(rep('homeadv',nrow(obs_test)),rownames(x))]
    x_home <- apply(matrix(c(x_home_player1,x_home_player2,x_home_player3,
                             x_home_player4,x_home_player5,x_home_player6,
                             x_home_player7,x_home_player8,x_home_player9,
                             x_home_player10,x_home_player11,x_home_player12),
                           ncol = 12),
                    1, mean,na.rm = TRUE)
    x_home[is.na(x_home)] <- 0

    x_away_player1 <- x[match(as.character(obs_test$away_team_player1),rownames(x)
        )]
    x_away_player2 <- x[match(as.character(obs_test$away_team_player2),rownames(x)
        )]
    x_away_player3 <- x[match(as.character(obs_test$away_team_player3),rownames(x)
        )]
    x_away_player4 <- x[match(as.character(obs_test$away_team_player4),rownames(x)
        )]
    x_away_player5 <- x[match(as.character(obs_test$away_team_player5),rownames(x)
        )]
    x_away_player6 <- x[match(as.character(obs_test$away_team_player6),rownames(x)
        )]
```

```
      x_away_player7 <- x[match(as.character(obs_test$away_team_player7),rownames(x)
          )]
      x_away_player8 <- x[match(as.character(obs_test$away_team_player8),rownames(x)
          )]
      x_away_player9 <- x[match(as.character(obs_test$away_team_player9),rownames(x)
          )]
      x_away_player10 <- x[match(as.character(obs_test$away_team_player10),rownames(
          x))]
      x_away_player11 <- x[match(as.character(obs_test$away_team_player11),rownames(
          x))]
      x_away <- apply(matrix(c(x_away_player1,x_away_player2,x_away_player3,
                               x_away_player4,x_away_player5,x_away_player6,
                               x_away_player7,x_away_player8,x_away_player9,
                               x_away_player10,x_away_player11),
                             ncol = 11),
                       1, mean,na.rm = TRUE)

      pmdiff <- x_home - x_away
      train_pm <- c(train_pm,pmdiff)
    }

    # Use information for OLR
    OLR <- polr(formula = 'result~pmdiff',
                data = data.frame(result = factor(train_results, levels = c('A','D',
                    'H'),ordered = TRUE),
                                  pmdiff = train_pm))


  }

}


score <- data.frame(observed = factor(observed, levels = c('H', 'D', 'A')),
                    win = Prob_home,
                    draw = Prob_draw,
                    loss = Prob_away)


saveRDS(object = score,file = 'PM_hvattum4y_new.rds')


#get scores

score <- readRDS('PM_hvattum4y_new.rds')
score$observed_win <- ifelse(score$observed == 'H', 1 , 0)
score$observed_draw <- ifelse(score$observed == 'D', 1 , 0)
score$observed_loss <- ifelse(score$observed == 'A', 1 , 0)

```

```
score$IgnoranceScore <- apply(score[,2:7], 1, IGN)
score$BrierScore <-apply(score[,2:7], 1, BrierScore)
score$RPS <- apply(score[,2:7], 1, RPS2)


mean(score$IgnoranceScore)
sd(score$IgnoranceScore)

mean(score$BrierScore)
sd(score$BrierScore)

mean(score$RPS/2)
sd(score$RPS/2)

nrow(score)

s <- score[-c(2405:2466),]

mean(s$IgnoranceScore)
sd(s$IgnoranceScore)

mean(s$BrierScore)
sd(s$BrierScore)

mean(s$RPS/2)
sd(s$RPS/2)


s<-  summary(OLR)
xtable(s$coefficients)
```

A.4 Code for the Poisson based models

A.4.1 Independent Poisson

```
##########################################################
# Ranking soccer teams on current strength: MLE approach
# C. Ley
##########################################################

# packages
{
setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/')
library(engsoccerdata)
library(tidyverse)
library(skellam)
library(caret)
```

```
source('BrierRpsIgnorance.R')
}

#DATA
{
# Premier League
PL_data <- england


# Data cleaning

# filter for right dataset
data <- PL_data %>%
  filter(tier == 1,
         Season %in% 2006:2017)

data %>% group_by(Season) %>%
  summarise(matches = length(Date),
            team = length(levels(factor(home))))


#make round parameter
id <- order(data$Season, data$Date)
data <- data[id,]
data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )
}

#likelihood function

indeppoiss.mle <- function(par, data){

  #parameters from data
  teams <- sort(levels(factor(data$home)))
  gi <- data$hgoal
  gj <- data$vgoal
  hteam <- as.character(data$home)
  vteam <- as.character(data$visitor)

  #restrictions
  c <- abs(par[1])
  h <- abs(par[2])
  strength <- par[3:(length(par))]
  last_strength <- -sum(strength)
  strength <- c(strength, last_strength )

  #features for likelihood
  ri <- strength[match(hteam, teams)]
  rj <- strength[match(vteam, teams)]

```

```
  lambda_i <- exp(c + (ri+h)-(rj))
  lambda_j <- exp(c + (rj)-(ri+h))

  date <- as.Date(data$Date)
  x <- as.numeric(max(as.Date(data$Date)) - date)
  halfperiod <- 360
  wtime <- 0.5 ^ (x/halfperiod)


  lambda_i_exp_gi <- apply(matrix(c(lambda_i,gi),ncol = 2), 1,function(a){return(a
      [1]**a[2])})
  rev_fact_gi <- 1/(factorial(gi))
  exp_neg_lambda_i <- exp(-lambda_i)

  lambda_j_exp_gj <- apply(matrix(c(lambda_j,gj),ncol = 2), 1,function(a){return(a
      [1]**a[2])})
  rev_fact_gj <- 1/(factorial(gj))
  exp_neg_lambda_j <- exp(-lambda_j)

  matrix <- matrix(c(lambda_i_exp_gi,
                     rev_fact_gi,
                     exp_neg_lambda_i,
                     lambda_j_exp_gj,
                     rev_fact_gj,
                     exp_neg_lambda_j),
                   ncol=6)

  densities <-  apply(matrix, MARGIN = 1,  FUN = prod)
  densities <- apply(matrix(c(densities,wtime),ncol = 2), 1,function(a){return(a[1]*
      *a[2])})
  return(-sum(log(densities)))

}


# 2008:max(season)

predict <- c()
observed <- c()

p_win <- p_draw <- p_loss  <- c()
HAD <- c('H','D', 'A')

for (season_id in 2008:max(data$Season)){

  # datasets

  train_data <-  data %>%
    filter(Season %in% (season_id-2):(season_id-1))
```

```r

train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))

print('training dataset')
print(train_data %>% group_by(as.character(Season)) %>% summarise(rounds = length(
    levels(factor(round)))))

test <- data %>% filter(Season == season_id,round > 5)

print(test %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)
    ))))

teams <- sort(unique(c(levels(factor(train_data$home)),levels(factor(train_data$
    visitor)))))

# initialize prediction

par <- c(1,1,rep(0,length(levels(factor(train_data$home)))-1))
par_optim <- optim(par = par,fn = indeppoiss.mle, data = train_data,gr = "BFGS")
par_optim

# predict

for (i in 6:max(test$round)){

  # 1. MLE based on updated train_data
  par_optim <- optim(par = par,fn = indeppoiss.mle, data = train_data,gr = "BFGS")

  # 2. Parameter extraction
  par <- par_optim$par

  c <- abs(par[1])
  h <- abs(par[2])

  strength <- par[3:(length(par))]
  last_strength <- -sum(strength)
  strength <- c(strength, last_strength)

  teams <- sort(unique(c(levels(factor(train_data$home)),levels(factor(train_data$
      visitor)))))


  # 3. Parameters from test_data (observed)
  t <- test %>% filter(round == i)
  gi <- t$hgoal
  gj <- t$vgoal
  hteam <- as.character(t$home)
  vteam <- as.character(t$visitor)

```

```
    # 4. Features
    ri <- strength[match(hteam, teams)]
    rj <- strength[match(vteam, teams)]

    lambda_i <- exp(c + (ri+h)-(rj))
    lambda_j <- exp(c + (rj)-(ri+h))


    # 5. Predict loss, draw, win
    loss <-  pskellam(q = -1,
                      lambda1 = lambda_i,
                      lambda2 = lambda_j)


    draw <- dskellam(x = 0,
                     lambda1 = lambda_i,
                     lambda2 = lambda_j)


    win <- 1-pskellam(q = 0,
                      lambda1 = lambda_i,
                      lambda2 = lambda_j)


    p_win <- c(p_win, win)
    p_draw <- c(p_draw, draw)
    p_loss <- c(p_loss, loss)
    # 6. Save predictions
    observed <- c(observed, as.character(t$result))

    # 7. Update train data
    train_data <- rbind(train_data, t)
    print(paste(season_id,i, ' '))
  }



}

score <- data.frame(observed = factor(observed, levels = c('H', 'D', 'A')),
                    win = p_win,
                    draw = p_draw,
                    loss = p_loss)
score

saveRDS(object = score,file = 'IndependentPoission_CL2019_MLE.rds')


#get scores
```

```

score <- readRDS('IndependentPoission_CL2019_MLE.rds')
score$observed_win <- ifelse(score$observed == 'H', 1 , 0)
score$observed_draw <- ifelse(score$observed == 'D', 1 , 0)
score$observed_loss <- ifelse(score$observed == 'A', 1 , 0)

score$IgnoranceScore <- apply(score[,2:7], 1, IGN)
score$BrierScore <-apply(score[,2:7], 1, BrierScore)
score$RPS <- apply(score[,2:7], 1, RPS2)

mean(score$IgnoranceScore)
mean(score$BrierScore)
mean(score$RPS)/2
sd(score$RPS/2)
```

A.4.2 Bivariate Poisson

```
##############################################################
# Ranking soccer teams on current strength: MLE approach
# C. Ley
##############################################################

# packages
{
setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/')
library(engsoccerdata)
library(tidyverse)
library(skellam)
library(caret)
source('BrierRpsIgnorance.R')
}

#Data
{
  # Premier League
  PL_data <- england


  # Data cleaning

  # filter for right dataset
  data <- PL_data %>%
    filter(tier == 1,
           Season %in% 2006:2017)

  data %>% group_by(Season) %>%
    summarise(matches = length(Date),
              team = length(levels(factor(home))))

```

```

  #make round parameter
  id <- order(data$Season, data$Date)
  data <- data[id,]
  data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )
}

#functions
{
  Sumation_function <- function(row, lambda_c){

    lambda_i <- row[1]
    lambda_j <- row[2]
    gi <- row[3]
    gj <- row[4]
    min_goals <- row[5]

    sumation <- 0
    for (k in 0:min_goals){
      sumation <- sumation + choose(gi,k)*choose(gj,k)*factorial(k)*((lambda_c/(
          lambda_i*lambda_j))**k)

    }
    return(sumation)
  }
  #likelihood function

  bivpoiss.mle <- function(par, data){

    #parameters from data
    teams <- sort(unique(levels(factor(data$home)),levels(factor(data$visitor))))
    gi <- data$hgoal
    gj <- data$vgoal
    hteam <- as.character(data$home)
    vteam <- as.character(data$visitor)

    #restrictions
    c <- abs(par[1])
    h <- abs(par[2])
    lambda_c <- abs(par[3])
    strength <- par[4:(length(par))]
    last_strength <- -sum(strength)
    strength <- c(strength, last_strength )

    #features for likelihood
    ri <- strength[match(hteam, teams)]
    rj <- strength[match(vteam, teams)]

    lambda_i <- exp(c + (ri+h)-(rj))
```

```
    lambda_j <- exp(c + (rj)-(ri+h))

    date <- as.Date(data$Date)
    x <- as.numeric(max(as.Date(data$Date)) - date)
    halfperiod <- 390
    wtime <- 0.5 ^ (x/halfperiod)


    lambda_i_exp_gi <- apply(matrix(c(lambda_i,gi),ncol = 2), 1,function(a){return(a
        [1]**a[2])})
    rev_fact_gi <- 1/(factorial(gi))

    lambda_j_exp_gj <- apply(matrix(c(lambda_j,gj),ncol = 2), 1,function(a){return(a
        [1]**a[2])})
    rev_fact_gj <- 1/(factorial(gj))
    exp_ijc <- exp(-(lambda_i+lambda_j+lambda_c))


    min_goals <- apply(matrix(c(gi,gj),ncol = 2), MARGIN = 1 , min)
    sumation_matrix <- matrix(c(lambda_i, lambda_j, gi , gj, min_goals), ncol = 5)
    sumations <- apply(sumation_matrix, MARGIN = 1, FUN = Sumation_function, lambda_
        c)


    matrix <- matrix(c(lambda_i_exp_gi,
                       rev_fact_gi,
                       lambda_j_exp_gj,
                       rev_fact_gj,
                       exp_ijc,
                       sumations),
                     ncol=6)

    densities <-  apply(matrix, MARGIN = 1,  FUN = prod)

    densities <- apply(matrix(c(densities,wtime),ncol = 2), 1,function(a){return(a
        [1]**a[2])})


    return(-sum(log(densities)))

  }


}




# 2008:max(season)
```

```r

predict <- c()
observed <- c()

p_win <- p_draw <- p_loss  <- c()
HAD <- c('H','D', 'A')

for (season_id in 2008:max(data$Season)){
  # datasets
  train_data <-  data %>%
    filter(Season %in% (season_id-2):(season_id-1))

  train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))

  print('training dataset')
  print(train_data %>% group_by(as.character(Season)) %>% summarise(rounds = length(levels(factor(round)))))

  test <- data %>% filter(Season == season_id,round > 5)

  print(test %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))

  # initialize prediction
  teams <- sort(unique(c(levels(factor(train_data$home)),levels(factor(train_data$visitor)))))

  par <- c(1,1,1,rep(0,length(levels(factor(train_data$home)))-1))
  par_optim <- optim(par = par,fn = bivpoiss.mle, data = train_data,gr = "BFGS")
  par <- par_optim$par
  par
  # predict

  for (i in 6:max(test$round)){

    # 1. MLE based on updated train_data
    par_optim <- optim(par = par,fn = bivpoiss.mle, data = train_data,gr = "BFGS")

    # 2. Parameter extraction
    par <- par_optim$par

    c <- abs(par[1])
    h <- abs(par[2])
    lambda_c <- par[3]
    strength <- par[4:(length(par))]
    last_strength <- -sum(strength)
    strength <- c(strength, last_strength)
    teams <- sort(unique(c(levels(factor(train_data$home)),levels(factor(train_data$visitor)))))
```

```


    # 3. Parameters from test_data (observed)
    t <- test %>% filter(round == i)
    gi <- t$hgoal
    gj <- t$vgoal
    hteam <- as.character(t$home)
    vteam <- as.character(t$visitor)

    # 4. Features
    ri <- strength[match(hteam, teams)]
    rj <- strength[match(vteam, teams)]

    lambda_i <- exp(c + (ri+h)-(rj))
    lambda_j <- exp(c + (rj)-(ri+h))


    # 5. Predict loss, draw, win
    loss <-  pskellam(q = -1,
                      lambda1 = lambda_i,
                      lambda2 = lambda_j)


    draw <- dskellam(x = 0,
                     lambda1 = lambda_i,
                     lambda2 = lambda_j)


    win <- 1-pskellam(q = 0,
                      lambda1 = lambda_i,
                      lambda2 = lambda_j)


    p_win <- c(p_win, win)
    p_draw <- c(p_draw, draw)
    p_loss <- c(p_loss, loss)
    # 6. Save prediction
    observed <- c(observed, as.character(t$result))

    # 7. Update train data
    train_data <- rbind(train_data, t)
    print(paste(season_id,i, ' '))
  }



}

score <- data.frame(observed = factor(observed, levels = c('H', 'D', 'A')),
```

```
                    win = p_win,
                    draw = p_draw,
                    loss = p_loss)
score


saveRDS(object = score,file = 'BivariatePoission_CL2019_MLE.rds')


#get scores

score <- readRDS('BivariatePoission_CL2019_MLE.rds')
score$observed_win <- ifelse(score$observed == 'H', 1 , 0)
score$observed_draw <- ifelse(score$observed == 'D', 1 , 0)
score$observed_loss <- ifelse(score$observed == 'A', 1 , 0)

score$IgnoranceScore <- apply(score[,2:7], 1, IGN)
score$BrierScore <-apply(score[,2:7], 1, BrierScore)
score$RPS <- apply(score[,2:7], 1, RPS2)


mean(score$IgnoranceScore)
sd(score$IgnoranceScore)

mean(score$BrierScore)
sd(score$BrierScore)

mean(score$RPS/2)
sd(score$RPS/2)
```

A.5 Code for the Weibull count model

A.5.1 Weibull count

```
##############################################################
# A bivariate Weibull count model for forecasting association football scores
# G. Boshnakov
##############################################################


# packages
{
  setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/'
      )
  library(engsoccerdata)
  library(Countr)
  library(skellam)
```

```
  library(caret)
  library(MASS)
  library(tidyverse)
  source('BrierRpsIgnorance.R')
}

# data
{
  # Premier League
  PL_data <- england
  PL_data%>% filter(tier == 1)

  data <- PL_data %>%
    filter(tier == 1,
           Season %in% 2006:2015)

  #make round parameter
  id <- order(data$Season, data$Date)
  data <- data[id,]
  data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )

}

# Frank copula function
{
  Frank.copula <- function(u,v,k){
    teller <- (exp(-k*u)-1)*(exp(-k*v)-1)
    noemer <- (exp(-k)-1)
    fraction <- teller/noemer
    copula <- -1/k*log(1+fraction)
    return(copula)

  }
}

# Cummulative density weibull function
{
  CDWeibull <- function(yi,shape, scale){

    if (yi < 0) {densities <- 0}
    else {
      seq <- c(0, seq_len(yi))
      densities <- dWeibullCount(seq,shape,scale)
      return(sum(densities))
    }

  }
}

```

```
# Likelihood function
{
  WeibullLL <- function(par, data){

    # parameters
    teams <- sort(unique(levels(factor(data$home)),levels(factor(data$visitor))))
    atk_teams <- par[1:length(teams)]# alpha's
    def_teams <- par[(length(teams)+1):(2*length(teams))] # beta's
    h <-par[(2*length(teams))+1] # home team advantage
    k <- par[(2*length(teams))+2]  # used for copula
    Ch <- par[(2*length(teams))+3]
    Ca <- par[(2*length(teams))+4]

    #parameters from data
    gi <- data$hgoal
    gj <- data$vgoal
    hteam <- as.character(data$home)
    vteam <- as.character(data$visitor)

    # restrictions ?

    # derivations
    atk_i <- atk_teams[match(hteam, teams)]
    def_i <- def_teams[match(hteam, teams)]
    lambda_i <- exp(atk_i+def_i+h)

    atk_j <- atk_teams[match(vteam, teams)]
    def_j <- def_teams[match(vteam, teams)]
    lambda_j <- exp(atk_j+def_j)

    # Time variety
    epsilon <- 1/500
    date <- as.Date(data$Date)
    x <- as.numeric(max(as.Date(data$Date)) - date)
    wtime <- exp(-epsilon*x)

    # Weibull densities
    MarginalDensHome <- apply(data.frame(x=gi,
                                         shape=Ch,
                                         scale=lambda_i),
                              MARGIN = 1,
                              FUN = function(a){return(dWeibullCount(x = a[1],shape
                                  = a[2],scale = a[3]))})



    MarginalDensAway <- apply(data.frame(x=gj,
                                         shape=Ca,
                                         scale=lambda_j),
```

```
                              MARGIN = 1,
                              FUN = function(a){return(dWeibullCount(x = a[1],shape = a[2],scale = a[3]))})

    # Cumulative densities

    F1 <- apply(data.frame(x=gi,
                           shape=Ch,
                           scale=lambda_i),
                      MARGIN = 1,
                      FUN = function(a){return(CDWeibull(yi = a[1],shape = a[2],scale = a[3]))})

    F2 <- apply(data.frame(x=gj,
                           shape=Ca,
                           scale=lambda_j),
                MARGIN = 1,
                FUN = function(a){return(CDWeibull(yi = a[1],shape = a[2],scale = a[3]))})

    Fm1 <-  apply(data.frame(x=gi-1,
                             shape=Ch,
                             scale=lambda_i),
                  MARGIN = 1,
                  FUN = function(a){return(CDWeibull(yi = a[1],shape = a[2],scale = a[3]))})

    Fm2 <- apply(data.frame(x=gj-1,
                            shape=Ca,
                            scale=lambda_j),
                     MARGIN = 1,
                     FUN = function(a){return(CDWeibull(yi = a[1],shape = a[2],scale = a[3]))})


    L <- Frank.copula(u = F1,v = F2, k = k) - Frank.copula(u = Fm1,v = F2, k = k) - Frank.copula(u = F1,v = Fm2, k = k) + Frank.copula(u = Fm1,v = Fm2, k = k)

    # own made up
    densities <- apply(matrix(c(L,wtime),ncol = 3), 1,prod)

    return(-sum(log(densities)))

  }


}
```

```
# Algorithm
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed_home <- c()
observed_draw <- c()
observed_away <- c()
for (season_id in 2008:2015){

  # Training data
  train_data <-  data %>%
    filter(Season %in% (season_id-2):(season_id-1))
  train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))
  print('training dataset')
  print(train_data %>% group_by(as.character(Season)) %>% summarise(rounds = length(
      levels(factor(round)))))

  # Testing data
  test <- data %>% filter(Season == season_id,round > 5)
  print(test %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)
      ))))

  # parameters
  teams <- sort(unique(levels(factor(train_data$home)),levels(factor(train_data$
      visitor))))
  atk_teams <- rep(0,length(teams)) # alpha's
  def_teams <- rep(0,length(teams)) # beta's
  h <- 0.2948 # home team advantage
  k <- -0.4561 # used for copula
  Ch <- 1.050
  Ca <- 0.9831

  par <- c(atk_teams,def_teams,h,k,Ch,Ca)

  par_optim <- optim(par = par,fn = WeibullLL, data = train_data,gr = "BFGS")

  for (round_id in 6:(max(test$round))){

    # Observation
    obs <- test %>% filter(round == round_id)

    # 1. MLE based on updated train_data
    par_optim <- optim(par = par,fn = WeibullLL, data = train_data,gr = "BFGS")

    # 2. Parameter extraction
    par <- par_optim$par
    teams <-  sort(unique(levels(factor(train_data$home)),levels(factor(train_data$
        visitor))))
    atk_teams <- par[1:length(teams)]# alpha's
```

```
def_teams <- par[(length(teams)+1):(2*length(teams))] # beta's
h <-par[(2*length(teams))+1] # home team advantage
k <- par[(2*length(teams))+2]  # used for copula
Ch <- par[(2*length(teams))+3]
Ca <- par[(2*length(teams))+4]

# 3. Parameters from test_data (observed)
gi <- obs$hgoal
gj <- obs$vgoal
hteam <- as.character(obs$home)
vteam <- as.character(obs$visitor)

# 4. Features
atk_i <- atk_teams[match(hteam, teams)]
def_i <- def_teams[match(hteam, teams)]
lambda_i <- exp(atk_i+def_i+h)

atk_j <- atk_teams[match(vteam, teams)]
def_j <- def_teams[match(vteam, teams)]
lambda_j <- exp(atk_j+def_j)


# 5. Predict loss, draw, win

# prob draw
for (i in 1:nrow(obs)){
  draw <- c()
  for (goal in 0:60){
    ProbForGoal <- dWeibullCount(x=goal,shape = Ch, scale = lambda_i[i])*
        dWeibullCount(x=goal,shape = Ca, scale = lambda_j[i])
    draw <- c(draw, ProbForGoal)
  }
  p_draw <- sum(draw)


  win <- c()
  for (goal in 0:60){
    ProbForGoal <- dWeibullCount(x=goal,shape = Ca, scale = lambda_j[i])*(1-
        CDWeibull(yi = goal, shape= Ch, scale = lambda_i[i]))
    win <- c(win, ProbForGoal)
  }
  p_win <- sum(win)

  loss <- c()
  for (goal in 0:60){
    ProbForGoal <- dWeibullCount(x=goal,shape = Ch, scale = lambda_i[i])*(1-
        CDWeibull(yi = goal, shape= Ca, scale = lambda_j[i]))
    loss <- c(loss, ProbForGoal)
  }
```

```
      p_loss <- sum(loss)

      if(sum(p_draw,p_win,p_loss)!=1){print('Sum probabilities is not 1')}

      # 6. Save predictions
      if (as.character(obs$result[i]) == 'H'){
        observed_home <- c(observed_home,1)
        observed_draw <- c(observed_draw,0)
        observed_away <- c(observed_away,0)

      }
      else {
        if (as.character(obs$result[i]) == 'A'){
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,0)
          observed_away <- c(observed_away,1)

        }
        else {
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,1)
          observed_away <- c(observed_away,0)

        }
      }


      Prob_home <- c(Prob_home,p_win)
      Prob_draw <- c(Prob_draw,p_draw)
      Prob_away <- c(Prob_away,p_loss)
    }
    # 7. Update train data
    train_data <- rbind(train_data, obs)
    print(paste(season_id,round_id, ' '))
  }
}


out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away)

all.equal(apply(out[,1:3],1,sum),rep(1,nrow(out)))

out$ignorance <- apply(out[,1:6], 1, IGN)
out$brierScore <-apply(out[,1:6], 1, BrierScore)
```

```
out$RPS <- apply(out[,1:6], 1, RPS2)


mean(out$ignorance)
sd(out$ignorance)

mean(out$brierScore)
sd(out$brierScore)

mean(out$RPS/2)
sd(out$RPS/2)

saveRDS(object = out,file = 'BivariateWeibullCountCopula.rds')
```

A.6 Code for the machine learning models

A.6.1 Random forest Baboota

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')
import itertools
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import interp
from pprint import pprint
from itertools import cycle
import matplotlib.pyplot as plt
from collections import OrderedDict
from sklearn.externals import joblib
from xgboost.sklearn import XGBClassifier
#from TrainTestSplit import TrainTestSplit
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import classification_report
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import learning_curve
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.impute import SimpleImputer


np.set_printoptions(precision = 10)

# load data
path = "C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/Predicting-English-Premier-Results-master/EngineeredData.csv"
data = pd.read_csv(path)

data.head(5)

# Change Season

vector = []
for i in range(len(data['Season'])):
    vector.append(data['Season'][i][0:4])
data['Season'] = vector

# Make Round Feature
data = data.sort_values(by=['Date'])

roundsperseason = np.repeat([i for i in range(1,39)],10)

listofdata = []
for season_id in sorted(set(data['Season'])):
    test = data.copy()
    test = test[test['Season'] == season_id]
    test['Round'] = roundsperseason
    listofdata.append(test)

data = pd.concat(listofdata)
data = data.sort_values(by=['Date'])

# Features needed
A = ["ATGD", "HTGD", "ACKPP", "HCKPP", "AGKPP", "HGKPP", "ASTKPP", "HSTKPP", "ASt", "HSt", "AStWeighted", "HStWeighted", "AForm", "HForm", "AOverall", "HOverall", "AAttack", "HAttack", "AMidfield", "HMidfield", "ADefense", "HDefense"]
B =  ["GD", "CKPP", "GKPP", "STKPP", "Streak", "WeightedStreak", "Form", "Overall", "Attack", "Midfield", "Defense"]

frames = []
for season_id in range(2008,2016):

    #1. make train and test
    train = data[(data['Season'] == str(season_id-2)) | (data['Season'] == str(season_id-1))].copy()
    firstrounds = data[(data['Season'] == str(season_id)) & (data['Round'] <= 5)].copy()
```

```
    train = pd.concat([train,firstrounds])
    test = data[(data['Season'] == str(season_id)) & (data['Round'] > 5)].copy()

    for match_id in test.index:

        #1. define match to predict
        obs = test[test.index == match_id].copy()

        #2. SPlit in x and y
        train_x = train[A+B]
        train_y = train['FTR']
        test_x = obs[A+B]
        test_y = obs['FTR']

        train_x = train_x.fillna(0)
        test_x = test_x.fillna(0)

        #3. define classiffier
        Classifier = RandomForestClassifier(bootstrap=True,
                                            class_weight=None,
                                            criterion='gini',
                                            max_depth=None,
                                            max_features='log2',
                                            max_leaf_nodes=None,
                                            min_impurity_split=1e-07,
                                            min_samples_leaf=2,
                                            min_samples_split=100,
                                            min_weight_fraction_leaf=0.0,
                                            n_estimators=150,
                                            n_jobs=-1,
                                            oob_score=True,
                                            random_state=None,
                                            verbose=0,
                                            warm_start=False)

        #4. Train and test model
        Classifier.fit(train_x,train_y)

        #5. Predict
        YPred = Classifier.predict(test_x)
        classLabels = ['H','A','D']
        Yprob = Classifier.predict_proba(test_x)

        #6.output
        probs = pd.DataFrame(Yprob, columns=['p_loss','p_draw','p_win'])
        probs['Observed'] = YPred
        frames.append(probs)

        #7. update train
```

```
        train = train.append(obs)
out = pd.concat(frames)
out.to_csv("C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/RFbaboota.csv",
    index=False)
```

A.6.2 Gradient boosting

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')
import itertools
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import interp
from pprint import pprint
from itertools import cycle
import matplotlib.pyplot as plt
from collections import OrderedDict
from sklearn.externals import joblib
from xgboost.sklearn import XGBClassifier
#from TrainTestSplit import TrainTestSplit
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import classification_report
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import learning_curve
from sklearn.model_selection import cross_val_score


np.set_printoptions(precision = 10)


# In[2]:


# load data
path = "C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/Predicting-English-
    Premier-Results-master/EngineeredData.csv"
data = pd.read_csv(path)

data.head(5)
```

```


# ## Data engineering
#

# In[3]:


# Change Season

vector = []
for i in range(len(data['Season'])):
    vector.append(data['Season'][i][0:4])
data['Season'] = vector


# In[4]:


# Make Round Feature
data = data.sort_values(by=['Date'])

roundsperseason = np.repeat([i for i in range(1,39)],10)

listofdata = []
for season_id in sorted(set(data['Season'])):
    test = data.copy()
    test = test[test['Season'] == season_id]
    test['Round'] = roundsperseason
    listofdata.append(test)

data = pd.concat(listofdata)
data = data.sort_values(by=['Date'])


# In[5]:


# Features needed
A = ["ATGD", "HTGD", "ACKPP", "HCKPP", "AGKPP", "HGKPP", "ASTKPP", "HSTKPP", "ASt",
    "HSt", "AStWeighted", "HStWeighted", "AForm", "HForm", "AOverall", "HOverall", "
    AAttack", "HAttack", "AMidfield", "HMidfield", "ADefense", "HDefense"]
B =  ["GD", "CKPP", "GKPP", "STKPP", "Streak", "WeightedStreak", "Form", "Overall",
    "Attack", "Midfield", "Defense"]


# In[7]:


```

```
season_id = 2008

#1. make train and test
train = data[(data['Season'] == str(season_id-2)) | (data['Season'] == str(season_id
    -1))].copy()
firstrounds = data[(data['Season'] == str(season_id)) & (data['Round'] <= 5)].copy()
train = pd.concat([train,firstrounds])
test = data[(data['Season'] == str(season_id)) & (data['Round'] > 5)].copy()


# In[11]:


test


# In[10]:


test['Round']


# In[20]:


frames = []
rounds = []
seasons = []

for season_id in range(2008,2016):

    #1. make train and test
    train = data[(data['Season'] == str(season_id-2)) | (data['Season'] == str(
        season_id-1))].copy()
    firstrounds = data[(data['Season'] == str(season_id)) & (data['Round'] <= 5)].
        copy()
    train = pd.concat([train,firstrounds])
    test = data[(data['Season'] == str(season_id)) & (data['Round'] > 5)].copy()

    #train = train[['FTR']+B].dropna(axis=0)
    #test = test[['FTR']+B].dropna(axis=0)

    for match_id in test.index:

        #1. define match to predict
        obs = test[test.index == match_id].copy()
        rounds.append(int(obs['Round']))
        seasons.append(int(obs['Season']))

```

```

        #2. SPlit in x and y
        train_x = train[B]
        train_y = train['FTR']
        test_x = obs[B]
        test_y = obs['FTR']


        #3. define classiffier
        Classifier = XGBClassifier(base_score=0.5,
                                   booster='gbtree',
                                   colsample_bylevel=1,
                                   colsample_bytree=1,
                                   gamma=0.2,
                                   learning_rate=0.05,
                                   max_delta_step=0,
                                   max_depth=3,
                                   min_child_weight=1,
                                   n_estimators=100,
                                   n_jobs=-1,
                                   nthread=None,
                                   objective='multi:softprob',
                                   random_state=0,
                                   reg_alpha=0,
                                   reg_lambda=0.6,
                                   scale_pos_weight=1,
                                   seed=None,
                                   silent=True,
                                   subsample=1)

        #4. Train and test model
        Classifier.fit(train_x,train_y)

        #5. Predict
        YPred = Classifier.predict(test_x)
        classLabels = ['H','A','D']
        Yprob = Classifier.predict_proba(test_x)

        #6.output
        probs = pd.DataFrame(Yprob, columns=['p_loss','p_draw','p_win'])
        probs['Observed'] = YPred
        frames.append(probs)

        #7. update train
        train = train.append(obs)
out = pd.concat(frames)
out.to_csv("C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/Xgboost_dropna.
    csv",index=False)

```

```

# In[22]:


out['season'] = seasons
out['round'] = rounds

out.to_csv("C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/
    Xgboostevolution.csv",index=False)


# In[40]:


importances = Classifier.feature_importances_
indices = np.argsort(importances)[::-1]
names = [B[i] for i in indices]

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.barh(range(train_x.shape[1]), importances[indices])

# Add feature names as x-axis labels
plt.yticks(range(train_x.shape[1]), names)

# Show plot
plt.show()


# In[44]:


names


# In[43]:


pd.DataFrame(zip(names,importances), columns=['feature','importance']).to_csv("C:/
    Users/Thiebe/Documents/academie jaar 2019-2020/thesis/Xgboostfeatureimportance.
    csv",index=False)
```

A.7 Code for the hybrid models

A.7.1 Random forest Groll

```
# hybrid random forest data

# packages
{
  setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/')
  library(engsoccerdata)
  library(tidyverse)
  library(mlr)
  library(skellam)
  library(Hmisc)
  library(party)
  source('BrierRpsIgnorance.R')

  substrRight <- function(x, n){
    substr(x, nchar(x)-n+1, nchar(x))
  }
}

# Data
{
  # basic data (outcomes)
  {
    # Premier League
    PL_data <- england

    data <- PL_data %>%
      filter(tier == 1,
             Season %in% 2006:2015)

    #make round parameter
    id <- order(data$Season, data$Date)
    data <- data[id,]
    data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )

    # data
    data$Date <- as.Date(data$Date)
    data$key <- paste0(data$Date,substr(as.character(data$home),1,1),substr(as.character(data$visitor),1,1),data$hgoal,data$vgoal)
    droplevels(data$home)
    droplevels(data$visitor)
  }

  # elo
```

```
{
  elo_data <- readRDS('elo_data.rds')
  elo_data$Date <- as.Date(elo_data$Date)
  elo_data$Season <- NULL
  merged <- merge(data, elo_data,by = c('Date','home','visitor'))

}


# age check why this doesnt work!
{
  agedata <- read.csv('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/
      models/EPL_player_age_data.csv')
  agedata$dob <- as.Date(agedata$dob, format = '%d/%m/%Y')
  agedata$startseason <- as.Date(agedata$startseason)
  agedata$age <- agedata$startseason - agedata$dob
  agedata$age <- as.numeric(agedata$age)/365.25
  agedata$season <- as.numeric(agedata$season)
  agedata <- agedata %>% group_by(season,club) %>% summarise(mean_age = mean(age))
  agedata$club <- str_remove(agedata$club,' FC')
  agedata$club <- str_remove(agedata$club,' AFC')
  colnames(agedata) <- c('Season','home', 'homeage')
  merged <- merge(merged,agedata,by =  c('Season','home'))
  merged$homeage <- with(merged, impute(homeage, mean))
  colnames(agedata) <- c('Season','visitor', 'awayage')
  merged <- merge(merged,agedata,by =  c('Season','visitor'))
  merged$awayage <- with(merged, impute(awayage, mean))


}

# fifa
{
  fifa <- read.csv("C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/
      Predicting-English-Premier-Results-master/EngineeredData.csv")
  fifa <- fifa[,c('Date','HomeTeam','AwayTeam',
                  'HAttack','HDefense','HMidfield','HOverall' ,
                  'AAttack','ADefense','AMidfield','AOverall' )]
  #fifa$Season <- substr(fifa$Season,start = 6,stop = 9)
  colnames(fifa)<- c('Date','home','visitor',
                     'HAttack','HDefense','HMidfield','HOverall' ,
                     'AAttack','ADefense','AMidfield','AOverall' )


  fifa$home <- str_remove(fifa$home,' FC')
  fifa$home <- str_remove(fifa$home,' AFC')
  fifa$home[fifa$home == 'Spurs'] <- 'Tottenham Hotspur'

  fifa$visitor <- str_remove(fifa$visitor,' FC')
```

```
  fifa$visitor <- str_remove(fifa$visitor,' AFC')
  fifa$visitor[fifa$visitor== 'Spurs'] <- 'Tottenham Hotspur'


  merged$k <- paste0(merged$Date,
                     substr(str_remove(merged$home,'AFC '),1,1),
                     substr(str_remove(merged$visitor,'AFC '),1,1))

  fifa$k <- paste0(fifa$Date,
                   substr(str_remove(fifa$home,'AFC '),1,1),
                   substr(str_remove(fifa$visitor,'AFC '),1,1))

  fifa$Date <- fifa$home <- fifa$visitor <- NULL


  merged <- merge(merged,fifa,by=c('k'))

  #paste0(merged$Date,merged$home,merged$visitor)[! merged$k %in% fifa$k]
  merged$k<- NULL

}


# oddset
{
  Oddset <- readRDS('Oddset.rds')
  Oddset$home <- Oddset$visitor <- Oddset$Date <- NULL
  merged <- merge(merged, Oddset,by = c('key'))
  merged$key <- NULL

}





# final Data set
{
  data <- data.frame(result = merged$result,
                     hgoal = merged$hgoal,
                     vgoal = merged$vgoal,
                     elo = merged$elo_home-merged$elo_away,
                     age = merged$homeage - merged$awayage,
                     betHome = merged$B365H,
                     betDraw = merged$B365D,
                     betAway = merged$B365A,
                     round = merged$round,
                     Season = merged$Season,
                     FifaAttack = merged$HAttack - merged$AAttack,
```

```
                        FifaMidfield = merged$HMidfield - merged$AMidfield,
                        FifaDefense = merged$HDefense - merged$ADefense,
                        FifaOVerall = merged$HOverall - merged$AOverall,
                        date = as.Date(merged$Date)
    )
    print(data %>%  group_by(as.character(Season)) %>% summarise(rounds = length(
        round)))

    id <- order(data$date)
    data <- data[id,]
  }
}

#Algorithm
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed_home <- c()
observed_draw <- c()
observed_away <- c()
for (season_id in 2008:2015){

  # Training data
  {
    train_data <-  data %>%
      filter(Season %in% (season_id-2):(season_id-1))
    train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5)
        )
    print('training dataset')
    print(train_data %>% group_by(as.character(Season)) %>% summarise(rounds =
        length(levels(factor(round)))))
  }

  # Testing data
  {
    test <- data %>% filter(Season == season_id,round > 5)
    print(test %>% group_by(Season) %>% summarise(rounds = length(levels(factor(
        round)))))
  }

  # Hyperparameters
  {
    B <- 5000



  }


```

```
for (round_id in 6:(max(test$round))){


  print(paste(season_id,' ',round_id))


  # Observation
  obs <- test %>% filter(round == round_id)
  train_data$age <- as.numeric(train_data$age)
  obs$age <- as.numeric(obs$age)

  # 1. Train Random Forest

  cf <- cforest(formula = hgoal + vgoal ~ elo + age +betHome  + betDraw + betAway
      +FifaAttack + FifaMidfield + FifaDefense + FifaOVerall,
                data = train_data,
                controls = cforest_control(ntree = B))

  # 3. Predict goal difference test
  observed_goals <- obs[,c('hgoal','vgoal')]
  colnames(observed_goals) <- c('hgoal_observed','vgoal_observed')
  predicted_goals <- predict(cf, newdata = obs, type = "response")
  predicted_goals <- matrix(unlist(predicted_goals),ncol = 2,byrow = TRUE)
  colnames(predicted_goals) <- c('hgoal_predicted','vgoal_predicted')
  cbind(observed_goals,predicted_goals)


  # 4. Predict win draww loss with skellam
  p_loss <-  pskellam(q = -1,
                      lambda1 = predicted_goals[,1],
                      lambda2 = predicted_goals[,2])


  p_draw <- dskellam(x = 0,
                     lambda1 = predicted_goals[,1],
                     lambda2 = predicted_goals[,2])


  p_win <- 1-pskellam(q = 0,
                      lambda1 = predicted_goals[,1],
                      lambda2 = predicted_goals[,2])


  Prob_home <- c(Prob_home,p_win)
  Prob_draw <- c(Prob_draw,p_draw)
  Prob_away <- c(Prob_away,p_loss)

  for (i in 1:nrow(obs)){
    if (as.character(obs$result[i]) == 'H'){
```

```
        observed_home <- c(observed_home,1)
        observed_draw <- c(observed_draw,0)
        observed_away <- c(observed_away,0)

      }
      else {
        if (as.character(obs$result[i]) == 'A'){
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,0)
          observed_away <- c(observed_away,1)

        }
        else {
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,1)
          observed_away <- c(observed_away,0)

        }
      }
    }

    # 7. Update train data
    train_data <- rbind(train_data, obs)
  }


}

out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away)

all.equal(apply(out[,1:3],1,sum),rep(1,nrow(out)))

out$ignorance <- apply(out[,1:6], 1, IGN)
out$brierScore <-apply(out[,1:6], 1, BrierScore)
out$RPS <- apply(out[,1:6], 1, RPS2)


mean(out$ignorance)
sd(out$ignorance)

mean(out$brierScore)
sd(out$brierScore)

mean(out$RPS/2)
```

```
sd(out$RPS/2)

saveRDS(object = out,file = 'RFgroll.rds')
```

A.7.2 Hybrid random forest

```
# hybrid random forest data

# packages
{
  setwd('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/models/R scripts/'
      )
  library(engsoccerdata)
  library(tidyverse)
  library(mlr)
  library(skellam)
  library(Hmisc)
  library(party)
  source('BrierRpsIgnorance.R')

  substrRight <- function(x, n){
    substr(x, nchar(x)-n+1, nchar(x))
  }
}

# Data
{
  # basic data (outcomes)
  {
    # Premier League
    PL_data <- england

    data <- PL_data %>%
      filter(tier == 1,
             Season %in% 2006:2015)

    #make round parameter
    id <- order(data$Season, data$Date)
    data <- data[id,]
    data$round <- rep(sort(rep(1:38,10)), max(data$Season)- min(data$Season) +1 )

    # data
    data$Date <- as.Date(data$Date)
    data$key <- paste0(data$Date,substr(as.character(data$home),1,1),substr(as.
        character(data$visitor),1,1),data$hgoal,data$vgoal)
    droplevels(data$home)
    droplevels(data$visitor)
  }

```

```
# elo
{
  elo_data <- readRDS('elo_data.rds')
  elo_data$Date <- as.Date(elo_data$Date)
  elo_data$Season <- NULL
  merged <- merge(data, elo_data,by = c('Date','home','visitor'))

}

# abilities
{
  ability_data <- readRDS('ability_data_final.rds')
  colnames(ability_data) <-c("Date","Season","home","visitor", "ability_home", "
      ability_away")
  ability_data$Date <- as.Date(ability_data$Date)
  ability_data$Season<- NULL
  merged <- merge(merged, ability_data,by = c('Date','home','visitor'))

}

# age check why this doesnt work!
{
  agedata <- read.csv('c:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/
      models/EPL_player_age_data.csv')
  agedata$dob <- as.Date(agedata$dob, format = '%d/%m/%Y')
  agedata$startseason <- as.Date(agedata$startseason)
  agedata$age <- agedata$startseason - agedata$dob
  agedata$age <- as.numeric(agedata$age)/365.25
  agedata$season <- as.numeric(agedata$season)
  agedata <- agedata %>% group_by(season,club) %>% summarise(mean_age = mean(age))
  agedata$club <- str_remove(agedata$club,' FC')
  agedata$club <- str_remove(agedata$club,' AFC')
  colnames(agedata) <- c('Season','home', 'homeage')
  merged <- merge(merged,agedata,by =  c('Season','home'))
  merged$homeage <- with(merged, impute(homeage, mean))
  colnames(agedata) <- c('Season','visitor', 'awayage')
  merged <- merge(merged,agedata,by =  c('Season','visitor'))
  merged$awayage <- with(merged, impute(awayage, mean))


}

# fifa
{
  fifa <- read.csv("C:/Users/Thiebe/Documents/academie jaar 2019-2020/thesis/
      Predicting-English-Premier-Results-master/EngineeredData.csv")
  fifa <- fifa[,c('Date','HomeTeam','AwayTeam',
                  'HAttack','HDefense','HMidfield','HOverall' ,
                  'AAttack','ADefense','AMidfield','AOverall' )]
```

```
  #fifa$Season <- substr(fifa$Season,start = 6,stop = 9)
  colnames(fifa)<- c('Date','home','visitor',
                     'HAttack','HDefense','HMidfield','HOverall' ,
                     'AAttack','ADefense','AMidfield','AOverall' )


  fifa$home <- str_remove(fifa$home,' FC')
  fifa$home <- str_remove(fifa$home,' AFC')
  fifa$home[fifa$home == 'Spurs'] <- 'Tottenham Hotspur'

  fifa$visitor <- str_remove(fifa$visitor,' FC')
  fifa$visitor <- str_remove(fifa$visitor,' AFC')
  fifa$visitor[fifa$visitor== 'Spurs'] <- 'Tottenham Hotspur'


  merged$k <- paste0(merged$Date,
                     substr(str_remove(merged$home,'AFC '),1,1),
                     substr(str_remove(merged$visitor,'AFC '),1,1))

  fifa$k <- paste0(fifa$Date,
                   substr(str_remove(fifa$home,'AFC '),1,1),
                   substr(str_remove(fifa$visitor,'AFC '),1,1))

  fifa$Date <- fifa$home <- fifa$visitor <- NULL


  merged <- merge(merged,fifa,by=c('k'))

  #paste0(merged$Date,merged$home,merged$visitor)[! merged$k %in% fifa$k]
  merged$k<- NULL

}


# oddset
{
  Oddset <- readRDS('Oddset.rds')
  Oddset$home <- Oddset$visitor <- Oddset$Date <- NULL
  merged <- merge(merged, Oddset,by = c('key'))
  merged$key <- NULL

}




# final Data set
{
```

```
    data <- data.frame(result = merged$result,
                       hgoal = merged$hgoal,
                       vgoal = merged$vgoal,
                       elo = merged$elo_home-merged$elo_away,
                       ability = merged$ability_home-merged$ability_away,
                       age = merged$homeage - merged$awayage,
                       betHome = merged$B365H,
                       betDraw = merged$B365D,
                       betAway = merged$B365A,
                       round = merged$round,
                       Season = merged$Season,
                       FifaAttack = merged$HAttack - merged$AAttack,
                       FifaMidfield = merged$HMidfield - merged$AMidfield,
                       FifaDefense = merged$HDefense - merged$ADefense,
                       FifaOVerall = merged$HOverall - merged$AOverall,
                       date = as.Date(merged$Date)
    )
    print(data %>%  group_by(as.character(Season)) %>% summarise(rounds = length(round)))

    id <- order(data$date)
    data <- data[id,]
  }
}

#Algorithm
Prob_home <- c()
Prob_draw <- c()
Prob_away <- c()
observed_home <- c()
observed_draw <- c()
observed_away <- c()
for (season_id in 2008:2015){

  # Training data
  {
    train_data <-  data %>%
      filter(Season %in% (season_id-2):(season_id-1))
    train_data <- rbind( train_data, data %>% filter(Season == season_id,round <= 5))
    print('training dataset')
    print(train_data %>% group_by(as.character(Season)) %>% summarise(rounds = length(levels(factor(round)))))
  }

  # Testing data
  {
    test <- data %>% filter(Season == season_id,round > 5)
```

```
  print(test %>% group_by(Season) %>% summarise(rounds = length(levels(factor(round)))))
}

# Hyperparameters
{
  B <- 5000



}


for (round_id in 6:(max(test$round))){


  print(paste(season_id,' ',round_id))


  # Observation
  obs <- test %>% filter(round == round_id)
  train_data$age <- as.numeric(train_data$age)
  obs$age <- as.numeric(obs$age)

  # 1. Train Random Forest

  cf <- cforest(formula = hgoal + vgoal ~ elo + ability + age +betHome  + betDraw + betAway +FifaAttack + FifaMidfield + FifaDefense + FifaOVerall,
                data = train_data,
                controls = cforest_control(ntree = B))

  # 3. Predict goal difference test
  observed_goals <- obs[,c('hgoal','vgoal')]
  colnames(observed_goals) <- c('hgoal_observed','vgoal_observed')
  predicted_goals <- predict(cf, newdata = obs, type = "response")
  predicted_goals <- matrix(unlist(predicted_goals),ncol = 2,byrow = TRUE)
  colnames(predicted_goals) <- c('hgoal_predicted','vgoal_predicted')
  cbind(observed_goals,predicted_goals)


  # 4. Predict win draww loss with skellam
  p_loss <-  pskellam(q = -1,
                      lambda1 = predicted_goals[,1],
                      lambda2 = predicted_goals[,2])


  p_draw <- dskellam(x = 0,
                     lambda1 = predicted_goals[,1],
                     lambda2 = predicted_goals[,2])
```

```


    p_win <- 1-pskellam(q = 0,
                        lambda1 = predicted_goals[,1],
                        lambda2 = predicted_goals[,2])


    Prob_home <- c(Prob_home,p_win)
    Prob_draw <- c(Prob_draw,p_draw)
    Prob_away <- c(Prob_away,p_loss)

    for (i in 1:nrow(obs)){
      if (as.character(obs$result[i]) == 'H'){
        observed_home <- c(observed_home,1)
        observed_draw <- c(observed_draw,0)
        observed_away <- c(observed_away,0)

      }
      else {
        if (as.character(obs$result[i]) == 'A'){
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,0)
          observed_away <- c(observed_away,1)

        }
        else {
          observed_home <- c(observed_home,0)
          observed_draw <- c(observed_draw,1)
          observed_away <- c(observed_away,0)

        }
      }
    }

    # 7. Update train data
    train_data <- rbind(train_data, obs)
  }


}

out <- data.frame(pwin = Prob_home,
                  pdraw = Prob_draw,
                  ploss = Prob_away,
                  owin = observed_home,
                  odraw = observed_draw,
                  oloss = observed_away)

all.equal(apply(out[,1:3],1,sum),rep(1,nrow(out)))
```

```

out$ignorance <- apply(out[,1:6], 1, IGN)
out$brierScore <-apply(out[,1:6], 1, BrierScore)
out$RPS <- apply(out[,1:6], 1, RPS2)


mean(out$ignorance)
sd(out$ignorance)

mean(out$brierScore)
sd(out$brierScore)

mean(out$RPS/2)
sd(out$RPS/2)

saveRDS(object = out,file = 'hybridRF.rds')
```