The Faculty of Engineering, Science and Medicine
Aalborg University
Department of Computer Science

Title: Odds Assessment on Football Matches.

Research Unit: Machine Intelligence.

Project period: Dat6, 31st January, 2007 – 1st June, 2007

Project group: d623a

Participants:
Tobias Christensen
Rasmus Dencker Hansen

Supervisor:
Manfred Jaeger

Number of copies: 7

Number of pages: 144

Synopsis:

This report presents models for automatic assessment of probabilities for outcomes of football matches. The motivation is the enormous turnover in betting, hence the focus is on creating a probability assessor which has the same or a better performance than the bookmakers' when assessing the outcomes of football matches.
Two datasets are used: a detailed containing match events covering the last three years and a results-only covering the last six years. The models are only based on data available prior to the start of the matches, hence primarily data about the previous matches.
The probability assessors are built using k-Nearest Neighbor, decision trees, regression on goals and combinations of those made by ensemble methods. The evaluation of the probability assessors is made both by well known absolute and domain specific pair-wise scoring rules.
The models are evaluated against real-life odds. This evaluation is primarily for bookmaking and secondarily for gambling. The results indicate that it is difficult to make models which are better than the bookmakers' but that it is possible to almost match them.

Preface

The topic of this master thesis in the Machine Intelligence Group on the University of Aalborg is automatic assessment of probabilities for outcomes of football matches.

In making this project, we got help from certain individuals and companies which we would like to thank. These are:

- **Statman ApS & Kresten Buch**, provided detailed match statistics from the Danish SAS-league and expert domain knowledge.
- **NordicBet**, provided odds for the matches in the Danish SAS-league and expert domain knowledge.
- **Birthe Damborg,** provided mathematical, primarily regarding notation, and linguistic assistance.

This project is a continuation of our previous project, [CH07], hence some sections and chapters in this report stem from this. These are listed below where sections marked with an * are almost directly reused while the rest are based on experiences and partly text from the previous report.

- **Section 2.1** The Prediction Market and Odds
- **Sections 3.1*, 3.2*, and 3.4*** Quality of Detailed Match Data, Specification of Detailed Match Data, and Available Odds
- **Sections 4.1*, 4.2*, and 4.4*** Expected Value and Expected Loss, Mean, Variance, and Standard Deviation, and Distance Measures
- **Chapter 5** Data Preparation
- **Section 6.1** Scoring Rules for Probability Assessors
- **Chapter 8** k-Nearest Neighbor

The project is made by:

Tobias Christensen	Rasmus Dencker Hansen

Contents

Chapter 1

Introduction

This section will start with the motivation behind the project, then move on to state the goals of the project, and end with the strategy to achieve these goals.

1.1 Motivation

In this project, gambling is defined as:

wagering money or something of value on an event with an uncertain, yet clearly definable, outcome with the primary intent of winning additional valuables.

Gambling is recognized as one of the oldest forms of entertainment. Dice-like objects called Astragali have been found dating as far back as 40,000 years [Geo07]. The Chinese invented a tile game called Go and playing cards while the Greek soldiers, the Egyptians, and the Romans used dices. Furthermore there are numerous stories of gambling throughout history: in ancient Rome, Caligula confiscated the property of dead soldiers and prisoners to cover his gambling debts, and lawmakers decreed that all children were to be taught to gamble and throw dice. The first legal casino opened in 1765 in Baden, Germany. Since then, many different types of games have been introduced, e.g. blackjack, poker, baccarat, and roulette.

When modern sports was introduced, gambling was present. Modern sports gambling started with horse racing in England during the 1700s. Modern football was introduced in the late 1800s and games were played on a regular basis which attracted the gamblers' attention. Some gamblers took bets from several other gamblers at a fixed price set for the individual event which is the same as bookmaking. Since then, fixed-odds gambling has become the

standard gaming style for sports wagers. A sports wager could be on what the outcome of an event will be but there are a large amount of different and more detailed types of wagers. In fixed-odds gambling, the gambler knows the odds at which the stake is put such that he can calculate the return of the bet if he predicts correctly.

There are numerous traditional bookmakers, and they have an astonishingly high turnover. Some examples are Bet365 with 7.8 billion DKK per year, Sportingbet with 12 billion DKK per year, and Danske Spil with 17 billion DKK per year. Danske Spil is a Danish bookmaker whereas the others started in the UK but have moved elsewhere. Another kind of bookmaking, which has turned up in the last few years, is betting exchange. At such a bookmaker, the gamblers play against each other and the role of the bookmaker is to keep track of all the stakes. The largest of these betting exchanges is BetFair which has a weekly turnover of around 600 billion DKK Note that the number includes both bets that have been sold (laid) and bought (backed), and that most traditional bookmakers lay off their bets in the betting exchange in order to decrease their own risk [nor]. This implies that bookmakers which receive a high turnover on a particular outcome of an event typically will sell, lay off, this bet in a betting exchange such that their risk is reduced.

Bookmakers try to find the correct odds before a specific market is open which is done by a domain expert. Due to the high turnover, this part of bookmaking is crucial and small improvements can potentially be worth billions. A few decision support systems exist like BetRadar, a tool which keeps track of all the odds for each wager for all bookmakers, such that it can be instantaneously spotted if the odds on a wager are different from the market. Such a system is not useful for setting the initial odds and it is only influenced by the market, implying that there might be a better way to set the odds. Hence presently no decision support systems exist which help the bookmakers set the correct odds. This implies that if the bookmaker could obtain some help on how to set the odds before other bookmakers set their odds, they would have a slight advantage over the market and make more money.

Since it is possible for anybody to make a bet, a good model could also be used as a decision support system for anyone who wants to gamble. At the betting exchange markets, there are very high limits on how big a bet can be so if the model is sufficiently good, a lot of money could also be earned there.

1.2 Goals

The main goal of the project is to establish a model which outputs probability assessments on outcomes of football matches, i.e. a probability assessor. In tests, it must perform so well that it can be useful for a bookmaker. This probability assessor should be built based on historic data, implying that nothing is known about what will happen at the event in question. The data should be based on accurately definable quantitative measures.

A probability assessor is expected to be useful for a bookmaker as decision support, not to set the odds directly. The probability assessor needs to be at least as good as the bookmakers to make probability assessments which are accurate enough, preferably it should outperform the bookmakers in tests. This implies that a method, to calculate which probability assessor produces the most accurate probability assessments, must be found such that the accuracy of a probability assessor can be determined. Once the best probability assessor is identified, it must be verified that it is at least as good as some selected bookmakers, and the level of certainty that this is not by chance must be determined too.

Furthermore, an additional and less important goal is to determine whether the found probability assessor also performs well as a gambler. This implies that we also need to determine a method of calculating whether a given model performs well as a gambler.

1.3 Strategy

The problem encountered in this project is a machine learning problem, implying that a large dataset exists from which a model has to be learned. Firstly, the problem domain must be explored to establish which kinds of models would be suitable. Previous work mainly focuses on prediction but some of it also considers probability assessment, hence a solution to the stated problem should use some of the experience from those models. Hence, a search in the literature for previous attempts to solve this problem should be made.

Furthermore in order to make our own models, conventional learning algorithms, fitting the problem definition, must be searched for and those, which resemble the problem domain the most, should be selected.

To use the learning algorithms, some data is required and an understanding of the data and the data quality should be obtained. In addition to that, the data needs to be preprocessed such that it can be used in other models and by the selected learning algorithms. The data must be split so that the test results are reliable.

Once a number of models have been identified, both from the literature and from the conventional learning algorithms, these will be tested. This requires a scoring system such that it is possible to determine which types and settings are best and also whether any are able to perform comparable to the bookmakers. The comparison with the bookmakers also includes measurements of the probability of the different models being better than the bookmakers.

Lastly, the models will be tested in order to determine whether they can beat the bookmakers as gamblers.

Chapter 2

Problem Setting

The problem encountered in this project is how to automatically assess odds or probabilities, based on historic data, for future football matches the odds should compare favourably to bookmakers'. This implies that to understand the problem setting, it is important to understand the odds market and how odds are usually set. The odds market is based on people and companies gambling or trading bets mainly over the Internet. The first section presents an overview of odds calculations and betting in general and then presents the odds market and how odds assessment is normally made.

A related work section is also presented. It consists of some incentives for making a model which compares favourably with the bookmakers'. First, it presents how sports is normally characterised, namely by using simple statistics. Afterwards, a survey of models, similar to the one this project wants to develop, is presented.

2.1 The Prediction Market and Odds

This section starts out by introducing the equations behind the calculation of odds. After that, an overview of the odds market is presented and some of the mechanisms behind the odds assessments are explained based on these equations.

2.1.1 Odds Calculation

A given wager on a match has a number of possible outcomes and on each outcome there are odds. A bet is a stake placed by a gambler on an outcome. The wager will be declared settled after the match is played and the outcome of the wager is observed. If the match is canceled or rescheduled, e.g. due

to weather conditions, the wager will be cancelled. If $outcome = 1, \ldots, n$ are the possible outcomes, $P(outcome = i)$ is the probability that outcome i will happen. This implies that

$$\sum_{i=1}^{n} P(outcome = i|Settled) = 1 \tag{2.1}$$

All cancelled bets count as if they were never placed, implying that the placed stake is returned, hence $P(outcome = i|Settled) = P(outcome = i)$.

In order for the bookmakers to win over the gamblers, the bookmakers have a pay-back-percentage. This percentage indicates how much a gambler on average will get back per unit betted over a large amount of bets. This percentage usually lies between 0.90–0.95.

To represent the available odds to the gamblers, the bookmakers will choose an odds-style and calculate the odds from the probabilities. There are several ways to represent odds. In this report, the European odds style, also known as decimal odds, is chosen and these will be referred to simply as odds in the remainder of this report. If an outcome is set to odds 2.5, it implies that the gambler wins his stake times 2.5. Note that the winnings include the stake, such that the actual profit will be the winnings minus the stake. The odds are calculated by

$$O_i = \frac{b}{P(outcome = i)} \tag{2.2}$$

where O_i is the odds for outcome i and b is the pay-back-percentage.

As an example of how to calculate the outcome odds on a football match, consider a bookmaker with the probabilities for 1–X–2 as 0.5–0.3–0.2 and a pay-back-percentage of 1. Then the odds are calculated as: $1/0.5 = 2$, $1/0.3 = 3.33$, $1/0.2 = 5$, respectively. If the bookmaker wants to make money, he could e.g. have a pay-back-percentage of 0.9 instead. This would yield these other odds for the example: $0.9/0.5 = 1.8$, $0.9/0.3 = 3$, $0.9/0.2 = 4.5$.

The bookmaker can calculate how large the pay back to the gamblers will be based on some information about the wager. Let $S_{i,j}$ be the stake customer j places on outcome i, so that $\sum_j S_{i_0,j}$ is the sum of all stakes placed on outcome i_0 and $\sum_{i,j} S_{i,j}$ is the sum of all stakes placed on this wager. If outcome i_0 occurs, the percentage paid back, pb_{i_0}, will be as showed in (2.3).

$$pb_{i_0} = O_{i_0} \cdot \frac{\sum_j S_{i_0,j}}{\sum_{k,j} S_{k,j}} = \frac{b}{P(outcome = i_0)} \cdot \frac{\sum_j S_{i_0,j}}{\sum_{k,j} S_{k,j}} \tag{2.3}$$

$\frac{\sum_j S_{i_0,j}}{\sum_{k,j} S_{k,j}}$ is the same as the percentage of the stakes, measured by size, placed on outcome i_0. E.g., if the probability for an outcome is calculated to

$P(outcome = i_0) = 0.3$, the total stakes are $\sum_{k,j} S_{k,j} = 10,000,000$, the stakes on the given outcome is $\sum_j S_{i_0,j} = 3,200,000$, and the pay-back-percentage is 0.9, the percentage payed back will be:

$$pb_{i_0} = \frac{0.9}{0.3} \cdot \frac{3,200,000}{10,000,000} = 0.96$$

This implies that if outcome i_0 occurs, the bookmaker will have to pay 0.96 of the $10,000,000$ back to the customers.

The bookmaker will not know how the stakes are placed in advance, however, if they can be estimated (2.3) can be rewritten to find the pay-back-percentage which should be used to ensure a profit. A profit is obtained whenever the percentage paid back, pb_{i_0}, is below 1, regardless of the outcome. So for all outcomes if (2.4) holds, a gain for the bookmaker is guaranteed.

$$1 > \frac{b}{P(outcome = i_0)} \cdot \frac{\sum_j S_{i_0,j}}{\sum_{k,j} S_{k,j}} \Leftrightarrow P(outcome = i_0) \cdot \frac{\sum_{k,j} S_{k,j}}{\sum_j S_{i_0,j}} > b \quad (2.4)$$

Using the same values as in the prior example, even though the stakes in a real world situation would probably change when the pay-back-percentage changes, it is possible to find the highest pay-back-percentage possible still to achieve a profit:

$$0.3 \cdot \frac{10,000,000}{3,200,000} > b \Leftrightarrow 0.9375 > b$$

Provided that the stakes do not change, a pay-back-percentage below 0.9375 yields a profit for the bookmaker in the given scenario.

A bet, *bet*, on a wager is a triple $bet = (A, S, O)$, where A is the outcome of the event on which the gambler plays, S is the stake and O is the odds. If A is observed, the gambler's return is the stake times the odds, which implies that the gambler's net win is $(O-1) \cdot S$. The gambler loses the entire stake, if the outcome is different from A.

For a bet to make sense, the gambler must expect a profit. E.g., if the gambler thinks that the probability for home win is 0.5, a bookmaker thinks the probability is 0.48, and that bookmaker has a pay-back-percentage of 0.92 on that wager, the expected payback on a one unit bet is:

$$1 \cdot O_{home} \cdot 0.5 - 1 = \frac{b}{p_{home}} \cdot 0.5 - 1 = \frac{0.92}{0.48} \cdot 0.5 - 1 \approx 1.92 \cdot 0.5 - 1 = -0.042$$

This implies that the gambler will expect a loss, hence there is no reason for him to place a bet. In contrast, if the bookmaker sets the probability to 0.44:

$$1 \cdot O_{home} \cdot 0.5 - 1 = \frac{b}{p_{home}} \cdot 0.5 - 1 = \frac{0.92}{0.44} \cdot 0.5 - 1 \approx 2.09 \cdot 0.5 - 1 = 0.045$$

This results in an expected gain of 0.045 units per 1 unit bet, so the gambler should place a bet.

It is always possible to calculate the pay-back-percentage from the odds, because everything is known about the bet except the pay-back-percentage. To illustrate this, a case with three outcomes is considered. Since exactly one of the three outcomes must occur, the sum of the three probabilities is 1:

$$1 = P(outcome = 1) + P(outcome = 2) + P(outcome = 3) \tag{2.5}$$

The odds of the three outcomes are known and using (2.2), these can be rewritten:

$$O_i = \frac{b}{P(outcome = i)} \;\Leftrightarrow\; P(outcome = i) = \frac{b}{O_i} \tag{2.6}$$

Combining (2.5) and (2.6) yields (2.7):

$$1 = \frac{b}{O_1} + \frac{b}{O_2} + \frac{b}{O_3} \;\Leftrightarrow\; b = \frac{O_1 \cdot O_2 \cdot O_3}{O_1 \cdot O_2 + O_1 \cdot O_3 + O_2 \cdot O_3} \tag{2.7}$$

When the pay-back-percentage is known, the probabilities can easily be calculated using (2.6).

2.1.2 Dutch Book Argument and Arbitrage Bets

A dutch book is a bet where it is possible for the gambler to place money with a single bookmaker in a way which secures him a profit on the bet regardless of the outcome of the bet. This also implies that the bookmaker will lose money regardless of the outcome.

Definition 2.1 (Dutch Book Argument)**.** *Let $A_1, \ldots, A_n$ be all possible outcomes of the wager, and $O_1, \ldots, O_n$ be the odds for those outcomes. If $S_1, \ldots, S_n$ exists such that*

$$\min_{i=0}^{n} S_i \cdot O_i - \sum_{i=0}^{n} S_i > 0$$

the wager is called a dutch book.

Under normal circumstances, each bookmaker makes sure that this scenario does not happen. But the gambler may still sometimes be able to make arbitrage bets in scenarios where he places bets at more than one bookmaker.

Consider a bet with two possible outcomes, odds 1.9-2.05 at bookmaker A, and odds 2.05-1.9 at bookmaker B. If the gambler places the bet $(2, 100, 2.05)$ at bookmaker A and $(1, 100, 2.05)$ at bookmaker B, he wins $100 \cdot 2.05 - 2 \cdot 100 = 5$ regardless of the outcome. This scenario is costly for the bookmakers because the gamblers win. However, if one of the bookmakers is sure that he has the correct odds, he might be interested in this scenario because the gamblers will play more on such a wager. If this scenario happens often, some bookmakers will lose their money.

This scenario can be stopped by adjusting the odds and thus eliminating the sure bet. The adjustment can be done in two ways: either one bookmaker realises that his opening odds were wrong and then adjusts them towards the market, such that bookmaker A e.g. could adjust the odds to 1.95-1.95. Otherwise both bookmakers could adjust a little towards the middle, e.g. A to 1.95-2.0 and B to 2.0-1.95.

2.1.3 Practical Odds Assessment

A wager has three different states in its life cycle. Before it is presented to the gamblers, the bookmaker makes an odds assessment, based on expert knowledge and the form of the two teams. While the wager is available on the market, its odds change based on how the market reacts which can be spotted by a tool like BetRadar. When the game starts, the wager is normally taken off the market and in some cases, it is replaced by a live version of the same wager, the wager that is taken off is denoted the closing odds. All odds and live wagers besides the closing odds are ignored in this project. The final part of the cycle is the resulting which happens after the match is finished, where the winnings are paid back to the gamblers who won.

A number of international Internet bookmakers exist and it is possible to gamble at them from anywhere. This implies that sports betting works as a market where the bookmaker with the best odds often gets the customers. This further implies that the odds market could be compared to the stock market, in the sense that everybody can buy and sell bets anywhere which leads to the market adjusting itself. Note that this is only true for markets with a high turnover while markets with a low turnover typically do not adjust themselves as much. In practise, this implies that if one bookmaker has high odds on a home victory and the markets odds are high on away victory, the bookmaker with the different odds will not set the odds they believe are correct. Instead, they will set the odds from which they can obtain the highest potential profit. This will probably be setting home victory a little higher than the market and their odds for away win a little lower than the market. The reason being that gamblers will bet on the highest odds no matter the margin [nor].

Another consideration is whether the outcome of the wager is dependent on other types of wagers. An example could be the two types of wagers "Who will win the cup?" and "Who will win the match?", where the match is the cup final. If the turnover of the home team being cup winners prior in the tournament has been very large, and the turnover on the home team as the winner of the match also is very large, the bookmaker suffers a significant economic loss if the home team wins. To protect themselves against this scenario, the bookmakers will deliberately set low odds on e.g. home victory, such that the turnover decreases on this bet but increase on away victory, and in this way the risk is distributed [nor].

Another scenario which can change the odds is if the settings change, e.g. because of an injury on a key player. Because the information is available to both gamblers and bookmakers at the same time, it is important for the bookmakers to adjust their odds as quickly as possible. It may also be necessary to adjust the odds if one outcome is played much more than another.

Scenarios like these require a combination of domain specific expert knowledge, statistics on turnovers, odds, and other bookmakers but also the possibility of making real-life experiments with real bookmakers and real money. Since this is not possible in this project, we ignore these factors and instead focus on developing a model which is good at making probability assessments on football matches.

2.1.4 Profit Improvements

Since the bookmakers are companies, their owners wish to earn as much money as possible. This section presents ways of improving the bookmakers' profit.

There are several ways to get more customers. Advertisements make the customers aware of the bookmaker but at the cost of advertisement expenses. The odds can also be raised but it will result in a larger payback to all customers, including existing customers. More wagers on each match will also be interesting but it costs time and money to set the odds on these wagers and to result them.

Reducing the resulting time potentially increases the turnover because the customers are able to reuse their money more times. Due to the pay-back-percentage, the gamblers on average lose a little each time they play, so it is in the interest of the bookmaker to make it possible for the gamblers to play as often as possible.

The most difficult way of maximising profit is to assess the probabilities with higher accuracy but if this is achieved it can be very beneficial. When the probabilities are more accurate, the odds can be set higher which results in

more customers. Alternatively, odds can be lowered for some outcomes and raised on others in order to make the gamblers play on a specific outcome. This implies that it is very attractive to know the probability distribution for the match as accurately as possible.

The bookmakers try to set the correct odds. The probability assessment for the correct odds is the same as the distribution of outcomes on the bet if a large number of identical matches were played. The odds are decided by traders who are employed by the various bookmakers. The traders manually look at various statistics and try to set the correct odds based on that. These traders, although very skilled, can make mistakes due to the overwhelming amount of information and statistics available prior to a match. For that reason the probabilities can potentially be estimated more accurately using machine learning which will be done in this project.

2.2 Related Work

This section first presents different statements about sports and sports betting. Afterwards, we give an overview over different previous methods for both classification and probability assessment on sports events.

2.2.1 Motivation for Optimisations in Sports Betting

According to [nor] most probability assessments made before the market opens are based on expert analysis which again is based on a limited amount of statistics. Since there is no formal education, the experts are typically partly autodidact persons who have learned from others and by themselves. This also implies that little research has been made in the field of odds assessment.

The book *Moneyball* [Lew03] focuses on how the use of statistics in baseball has changed the game, and the reason why statistics is not just used by all sports participants right away.

As the following quote describes, the stock market as well as the gamblers and bookmakers in baseball are often too confident in their own abilities:

[Lew03, p.91] *Many people think they are smarter than others in the stock market – as if it's inert. Many people think they are smarter than others in baseball and that the game is simply what they think it is through their set of images/beliefs. Actual data from the market implies more than individual perception/belief. The same is true in baseball.*

If the same is true for football, some odds are biased towards what the experts believe instead of what they should be, hence there will be room for improvements.

In baseball, statistics is used increasingly but as the following quote illustrates, practical persons often do not understand nor want to understand statistics. Note that Paul (DePodesta) is a statistician employed by the Oakland Athletics and Billy (Beane) is the general manager:

[Lew03, pp.33-34] *It's what he doesn't say that is interesting. No one in big league baseball cares how often a college players walks; Paul cares about it more than just about anything else. He doesn't explain why walks are important. He doesn't explain that he has gone back and studied which amateur hitters made it in the big leagues, and which did not and why. He doesn't explain that the important traits in a baseball player were not all equally important. That foot speed, fielding ability, even raw power tended to be dramatically overpriced. That the ability to control the strike zone was the greatest indicator of future success. That the number of walks a hitter drew was the best indicator of whether he understood how to control the strike zone. Paul doesn't say that if a guy has a keen eye at the plate in college, he'll likely keep that keen eye in the pros. He doesn't explain that plate discipline might be an innate trait, rather than something a free-swinging amateur can be taught in the pros. He doesn't talk about all the other statistically based insights – the overwhelming importance of on-base percentage, the significance of pitchers seen per plate appearance – that he uses to value a hitter's contribution to a baseball offence. He doesn't stress the importance of generalising from a large body of evidence as opposed to a small one. He doesn't explain anything because Billy doesn't want him to. Billy was forever telling Paul that when you try to explain probability theory to baseball guys, you just end up confusing them.*

Very little statistics in used in football, according to [Fli02], AC Milan is one club that uses a little data mining. They collect data from workouts over a period of time and use this to prevent their players from obtaining serious injuries caused by stress fractures. Besides this case, the experts we have contacted are not aware if such statistics in being used, which in fact they believe it is not. This implies that some of the same success statistics has obtained in baseball might be obtainable in football as well.

2.2.2 Previous Models

[RS00] estimates the strengths of a team's offence and defence over time. This is done using Markov Chains Monte Carlo methods (MCMC). They use their model to determine which bets have an expected gain, such that it would be possible to determine which bets a gambler should take. Their gambling model takes the variance of the model into account, implying that a match where there is a high uncertainty on the outcome is given a large penalty on the expected outcome. They come to the conclusion that it is very hard to produce a working model which performs well as a gambler.

In [JFN06], Bayesian networks are used to model the probability that Tottenham Hotspur will win a given match. The prediction is based mainly on expert knowledge and simple facts as how good the opponent approximately is and whether any key players are missing. The results are promising but in order to be applied, they require expert knowledge of each individual team, implying that the approach is very hard to do for an entire league.

[DC97] is only based on the number of goals scored in previous matches and uses a Poisson distribution for each team to model the number of goals scored by each team. A maximum likelihood estimate is used to model the mean of the Poisson distributions. The article also states the following five criteria for a probability assessor:

- The model should take into account the different abilities of both teams in a match
- There should be allowance for the fact that teams playing at home generally have some advantage - "Home effect"
- The most reasonable measure of a team's ability is likely to be based on a summary measure of their recent performance
- The nature of football is such that a team's ability is likely to be best summarised in separate measures of their ability to attack (to score goals) and their ability to defend (to not concede goals)
- In summarising a team's performance by recent results, account should be taken of the ability of the teams that they have played against

The results are very promising, hence this model is studied in more detail in Chapter 9. The model is changed in [CDLR02] such that it is faster to compute. This report, however, only focuses on the original model.

[JI03] suggests an ordered probit regression model based on the goals scored in previous matches together with importance of game and geographical distance. The paper also presents an overview over the odds market and how to play against the bookmakers. The results are rather good and show a possible profit of 0.04 if it were used on actual matches. The model is similar to [DC97] and is not studied further.

Ensemble methods perform rather good in classification which is shown in [Die00b]. There is reason to belive that ensemble methods for probability assessment could be useful, hence this is studied in Chapter 10. Ensembles of expert opinions are also used in [DMP+06] in order to make probability assessments for American football matches.

It was not possible to find models using more advanced in-game statistics e.g. corner kicks, crosses, or shots on goal. This indicates that little research has been made and that there may be room for improvements in this area.

Chapter 3

Available Data

This chapter presents the data available for the machine learning process. We have received three kinds of data; detailed match data from Statman, results from Statman, and odds from NordicBet.

Detailed match data is available for the last three seasons of the top Danish league. The detailed match data is described in general in Section 3.1, including a description of how the quality in the data is ensured. A more thorough analysis is presented in Section 3.2 together with queries for some simple extractions.

The match result data contains more leagues and seasons and is described in detail in Section 3.3.

The available odds, and how they are matched up with the correct matches, is presented in Section 3.4.

3.1 Quality of Detailed Match Data

Accurate data for the problem domain is needed in order to get a satisfying result. The detailed match data used in this project is made by Statman. In this section, background information about the company, the way the data is made, and what the data contains is reviewed.

3.1.1 About Statman

Statman is a Danish company based in Copenhagen which produces football statistics for the Danish SAS league, the UEFA Champions League, and the FIFA World Cup. The aim of the company is to produce accurate data which can be used to compare players and teams. Statman uses the data for

a variety of products e.g. automatic graphic generation of match previews and post match statistics for newspapers, live presentations of the scores and other match actions on websites, and to help bookmakers with odds setting and bet resulting. Bet resulting is the job the bookmakers do after the end of a given match in order to calculate payouts. e.g., bookmakers need to know the exact number of corners in order to result a wager labelled over/under 8.5 corners.

Statman is led by Kresten Buch, whom is also our contact to the company. There are 12 part-time analysts employed to collect the match data and one part-time programmer who maintains the system and develops the technical part of the various products.

Besides providing us with the match data Statman's owner, Kresten Buch, also provides us with expert domain knowledge, which can be used to get ideas of which models to build and which features to use.

3.1.2 The Data Specifications

Currently, the database holds 771 matches, including matches used for training the Statman staff, the matches from the 2006 World Cup, the matches from the 06/07 season of Champions League, some international matches, the 04/05, 05/06, and 06/07 seasons of the Danish SAS-league, and various other single matches. In this project, only the matches from the Danish SAS-league are used, implying that there are currently 575 matches in the database which is used in this project. Only the matches from the Danish SAS-league are used since it is assumed that a match from the Danish SAS-league will differ from a match at the World Cup finals, e.g. because of other players and different schedule constraints.

In order to be able to deliver both fast and accurate data, two rounds of analysis of each game are made. During each game, an interface is used to create live data about the most significant actions in the game. These data can be shown live on a webpage and post match graphics can be made shortly after the end of the match. This is important if the game is played late and the statistics has to be in the newspaper the next morning. The detailed analysis can be made at any time, either during the match or later on. But since it is more detailed and precise, it takes about twice the time of the live analysis to make, implying that it will be available later than the live analysis. This data is used to make player profiles, next round previews, and bookmaker resulting. The detailed analysis does not reuse anything from the live analysis.

On the bottom level, the data is made up of different actions. A live action type might be "scored a goal" and a detailed action type "finished with right foot - goal - deflected by team mate". There are 503 different detailed action

types, though only around 350 are in use. 56 of the detailed action types are also used as live action types. An action on the pitch can both be represented in the live and detailed analysis. A typical game is made up of 70–250 live actions and 500–1,000 detailed actions. Note that off the ball actions, like a run by a player without the ball which is done in order to fix one or more opponents, are not registered. These runs can be spotted indirectly, e.g. because the finishing player is able to make a more precise shot if some opponent pressure is removed from him by one of his team mates who makes an off the ball move.

An action has one or two players connected with it, depending on what is most relevant. E.g., a goal scored on penalty only includes the player shooting but a substitution includes both players. On most actions in the live analysis, only one player is registered since it is hard for the analyst to accurately identify both players while the game is being played. For the same reason, the profiles of the individual players are built using the detailed analysis.

Each action has an attached time stamp. In the live analysis, a match clock is responsible for the time stamp which is possible because the live analysis and the live TV-signal by default are synchronised. The detailed time stamps are for minor actions grouped in five minute intervals, and for major actions, like "finished with right foot - goal - deflected by team mate" time stamped manually.

All live actions are also linked with a coordinate set representing the position on the pitch where the action occurred. The same is applicable for major actions in the detailed analysis.

Since the action types are low level, different categories are built on top of them. A category contains a number of action types, and each action type can be in more than one category. Because of the detailed level of the action types in the detailed data, it is possible to make a large number of categories, e.g. "goals with left foot", "goals scored by deflection", and "all goals". The categories are less useful in the live analysis, however a category in live analysis like "shots on target" contains both "shots on goal - saved" and "goals". There are 115 different detailed categories.

3.1.3 Data Quality Assurance

Since quality is of the essence, Statman has taken several measures to assure the quality of the data. The most important is to have an easy-to-use interface for the analysts. A screenshot of the detailed interface is shown in Figure 3.1 on the facing page. The main part of the interface shows the 22 starting players and the substitutes. Each player has a number of buttons, and a click on one of those buttons either brings forward a new set

of buttons or resuls in an action being saved in the dataset. Shirt numbers, player positions, and player names are shown for each player so that it is easy to locate the correct player for the action.

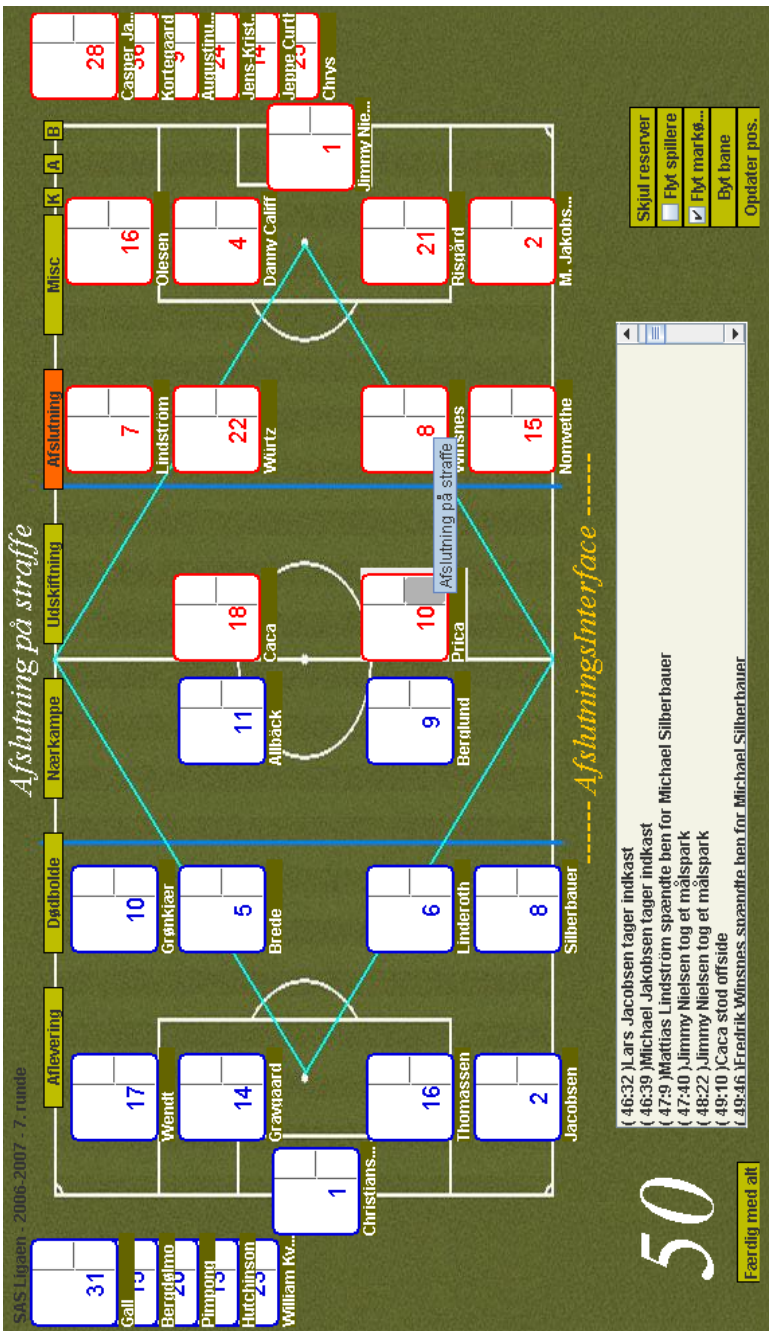

Figure 3.1: The Statman interface used for detailed analysis.

Before a new analyst analyses games, he receives training from an experi-

enced analyst. This training includes an introduction to the system and a number of training matches to analyse which are supervised by an experienced analyst.

To make the analysis smooth, the analyst is assigned a TV and a digital recorder such that it is possible to rewind and fast forward in the game as it is being played. This implies that important situations, such as goals, can be reviewed even during live analysis.

Since the live and detailed analysis are completely separated, the categories can be used to compare the key actions registered in the live and detailed analysis; this can e.g. be goals or corner kicks. When both the live and the detailed analysis are finished, the counts for each of the equivalent categories are be compared. If the number of actions is not the same, the analyst can locate the errors and correct them.

3.1.4 Selection of Data

Since this project aims at achieving the best possible probability assessments, the data with the best quality and the most details are used. This implies that only the matches on which the detailed analysis is completed are used, and the data from the live analysis are ignored for the remainder of this report.

3.2 Specification of Detailed Match Data

This section contains detailed information about the relevant tables in the dataset. Some columns are left out as they are not relevant while other columns and tables are renamed from Danish into English.

Only tables used in the detailed analysis are presented because all data analysis is performed on that part of the data as described in Section 3.1.4. A diagram of the database is shown in Figure 3.2.

Throughout the remainder of this report, the font style used for tables is `table` and `column` for columns.

This section presents the table structure and some data samples of the most important tables.

3.2.1 The `action` Table

The `action` table contains information about all actions in all matches. Table 3.1 on page 20 shows a sample of the table, i.e. some actions ordered by time for a single match with id 602.

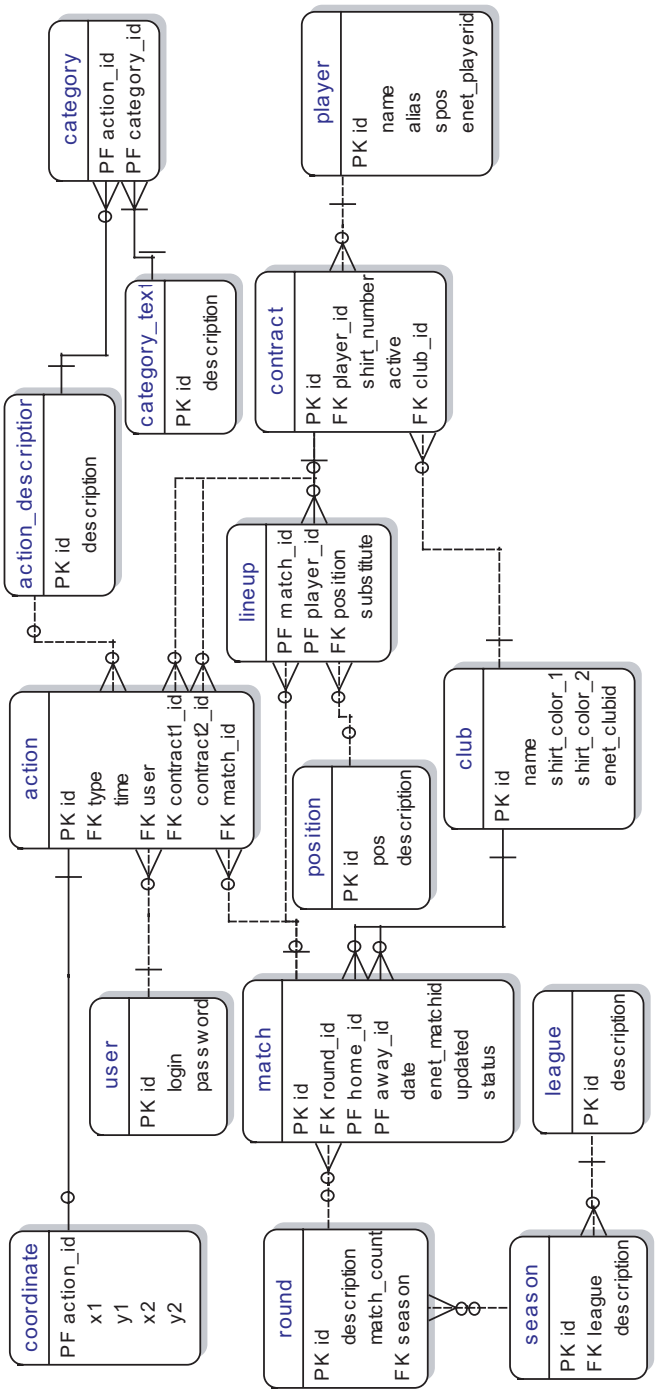

Figure 3.2: Selected parts of the Statman database.

id	contract1_id	contract2_id	match_id	type	user	time
284498	952	551	602	51	16	4347
284505	952	551	602	51	16	4364
284511	953	414	602	175	16	4378
284527	951	0	602	216	16	4481
284528	551	951	602	950	16	4481
284521	569	0	602	30	16	4500
284525	551	953	602	24	16	4500
284503	170	952	602	120	16	4500

Table 3.1: Raw sample of detailed `action` table.

id is the primary key and is used as a unique identifier for each action. contract1_id is the contract id, see Section 3.2.9 on page 28, of the first player who participates in the action. This can be the player who passes the ball, the player who shoots, and so on. contract2_id is the contract id of the second player in the action. This player may be the one who receives the pass, the goalkeeper who saves the ball, and so on. Note that contract2_id might be 0, implying no second player e.g. if the action is a finish off target.

match_id is the id of the match, see Section 3.2.4 on page 23. type is the action type, e.g. "took a throw in". The action type corresponds to a value in the `action_description` table, see Section 3.2.2. user is the id of the analyst who registered the action, which enables evaluation of the analysts afterwards.

time is the second where the action was made. Note that each half is 45 minutes long but may have extra time, so the use of the 46th minute is potentially ambiguous. To resolve this problem, the second $45 \cdot 60 = 2700$ is used for all actions which happen in the extra time in the first half and $90 \cdot 60 = 5400$ is used for all actions which happen in the extra time in the second half. As described in Section 3.1.2 on page 15, some actions are grouped in five minute intervals. In Table 3.1 actions 284521, 284525 and 284503 are examples of such. This implies that action 284521 may have happened before action 284498, even though it has a higher time stamp. Recall that it is possible to rewind and forward the match, such that the actions may not be registered in the order in which they happened.

Some actions are also linked with coordinates, however it is only the ones considered most important. The coordinates are stored in another table, `coordinate`. Even though the pitches on different stadiums have different sizes, one single size is assumed for simplicity. The entire pitch is 105×68 metres and it is divided into a number of small areas each 1×1 meter. The coordinates are measured according to each team, such that the coordinate (0,0) for the home team is measured starting from the left corner in their

end, and the coordinate (0,0) for the away team is measured from the left corner in their end. This implies that the coordinate (0,0) for each of the two teams is in opposite corners of the pitch. The coordinates (x1, y1) refers to the position of the player with contract1_id when he performed his part of the action and (x2, y2) refers to the position of the player with contract2_id when he performed his part of the action. The coordinates (x2, y2) can be (-1,-1), which indicates that contract2_id equals 0, implying that no second player is connected with the action. In Table 3.2, a sample of the `coordinate` table is presented.

actionid	x1	y1	x2	y2
284510	80	63	97	23
284517	104	0	97	35
284511	87	27	-1	-1
284527	82	39	-1	-1
284528	84	44	82	39

Table 3.2: Raw sample of the `coordinate` table.

3.2.2 The `action_description` Table

The available actions are described in the `action_description` table. A sample of the table is shown in Table 3.3.

id	description
24	makes an ordinary pass to offensive zone to
30	fails an ordinary pass to offensive zone to
51	takes a throw in to
120	tackled the ball to a team mate – tackled
175	tripped
216	finished with right foot – missed target
950	made a major keypass – ordinary flat pass

Table 3.3: Raw sample of the `action_description` table.

id is the primary key of the table, and the value referred to in the `action` table. description is a description of the action, which is both used during analysis and later when the data is presented.

3.2.3 The `category` and `category_text` Tables

Categories are made as aggregations of action types, as explained in Section 3.1.2 on page 15. Since a category may contain any number of action types and an action type can be in any number of categories, a table for this relationship is needed. That table is `category`, and a sample of the table is shown in Table 3.4. action_id is the action id, which corresponds to an id in the `action_description` table and is also used in the type column in the `action` table. category_id is the category id which corresponds to an id in the `category_text` table as explained below.

action_id	category_id
24	6
24	33
175	41
216	2
216	84
216	95

Table 3.4: Raw sample of the `category` table.

A sample of the `category_text` table is shown in Table 3.5. id is the id of the category and description is the description associated with it. A complete list of all categories can be found in the Appendix in Table A.1 on page 140.

id	description
1	Finished on target
2	Finished off target
6	Completed pass
33	Completed offensive short pass
41	Committed a misconduct
46	Finished on target with right foot
83	Finished on target with foot
84	Finished off target with foot
95	Finished off target with right foot in open play
97	Finished on target with right foot in open play

Table 3.5: Raw sample of the `category_text` table.

To illustrate the composition of categories, an example of different categories, which are subsets of each other, is shown in Table 3.6. The most

general category is category_id 1, "Finished on target". category_id 83, "Finished on target with foot", is a subset of "Finished on target" but does not contain finishes on target with head or other parts of the body. category_id 46, "Finished on target with right foot", contains half the actions of "Finished on target with foot" because all finishes with left foot are left out. category_id 97, "Finished on target with right foot in open play", has the same actions as "Finished on target with right foot" except for the finishes on set pieces.

category_id	actions
1	35, 58, 200, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 223, 224, 225, 226, 227, 228, 229, 230, 231, 250, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 273, 274, 275, 276, 277, 278, 279, 280, 281, 300, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 323, 324, 325, 326, 327, 328, 329, 330, 331, 350, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 373, 374, 375, 376, 377, 378, 379, 380, 381, 397, 398, 399, 423, 424, 425, 427, 430, 431, 432, 433, 434, 435, 437, 440, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 459, 460, 461, 462, 463, 470, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 489, 490, 491, 492, 493, 961, 990, 991, 992, 993, 994, 995
83	35, 58, 200, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 223, 224, 225, 226, 227, 228, 229, 230, 231, 250, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 273, 274, 275, 276, 277, 278, 279, 280, 281, 397, 398, 399, 423, 424, 425, 427, 430, 431, 432, 433, 434, 435, 437, 440, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 459, 460, 461, 462, 463, 470, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 489, 490, 491, 492, 493, 961, 990, 991, 992, 993, 994, 995
46	200, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 223, 224, 225, 226, 227, 228, 229, 230, 231, 397, 398, 399, 423, 424, 425, 427, 440, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 459, 460, 461, 462, 463, 990, 992, 994
97	200, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 223, 224, 225, 226, 227, 228, 229, 230, 231

Table 3.6: List of actions associated with different categories.

3.2.4 The `match` Table

The `match` table contains a row for each match which has either been analysed or which will be analysed in the near future. A sample of the `match`

table is shown in Table 3.7.

id	home	away	date	round	updated	status	enet_id
602	7	16	1154268000	130	1154279320	5	159011
614	4	10	1156035600	133	1156095045	5	159027
615	11	7	1156035600	133	1156094612	5	159025
623	19	11	1156640400	136	1157912024	5	159035
626	2	9	1156651200	136	1159044687	5	159031

Table 3.7: Raw sample of the `match` table.

The id column is the primary key in the table, and it is referred in the `action` table, see Section 3.2.1 on page 18, and the `lineup` table, see Section 3.2.7 on page 26.

home and away are the ids of the home team and the away team, corresponding to an entry in the `club` table, see Section 3.2.6 on page 26.

date is the date and time where the match will or did start, depending on whether the match has been played. The value is the number of seconds since 1 January 1970 00:00:00, which implies that 1154268000 corresponds to 30 July 2006 16:00:00. The updated column uses the same format and that value indicates when the match was last updated. An update of the match can e.g. be an added or changed action or a change in the lineup.

The round, which a match is part of, is referred by the round. The relationship between rounds, seasons, and leagues is described in Section 3.2.5 on the facing page.

status is the current status of the match. The values corresponds to:

0: Match not yet started.

1: Live analysis ongoing.

2: Live analysis interrupted (e.g. caused by a TV error).

3: Match finished.

4: Live analysis completed.

5: Detailed analysis completed. Match used in the learning data.

6: Detailed analysis completed. Match used in the test data.

Note that only matches where the detailed analysis is finished, status= 5 or status= 6, are used as described in Section 3.1.4 on page 18. Status level 6 is added in this project and is not used by Statman.

enet_id is another id for the match made by the company EnetPulse. enet_id is used to find equivalent matches from other datasets containing matches.

3.2.5 The `round`, `season`, and `league` Tables

The general structure of football is that every country has its own league system and that each league has a number of seasons. Each season consists of a number of rounds and each round of a number of matches. This is reflected in the database where each match is part of a round, each round is part of a season, and each season is part of a league.

A sample of the `round` table is presented in Table 3.8. id is the primary key of the table. description is the description of the round, which is typically the round number in the league. match_count is the number of matches in the round. That number helps the person who creates matches to assure that he did not forget any matches and that no matches are entered twice by accident. season is the season of which the round is a part.

id	description	match_count	season
5	24	6	1
36	16	6	2
118	2	2	23
130	3	6	35
133	6	6	35
136	7	6	35
144	9	6	35
149	3	16	36

Table 3.8: Raw sample of the `round` table.

A sample of the `season` table is presented in Table 3.9. id is the primary key of the table and referred in the `round` table. description is the description of the season. `league` is the id of the league of which the season is a part.

id	description	league
1	2003-2004	1
2	2004-2005	1
23	Gruppe A	16
35	2006-2007	1
36	2006-2007	19

Table 3.9: Raw sample of the `season` table.

A sample of the `league` table is presented in Table 3.10. id is the primary key of the table and referred in the `season` table. description is the description of the league.

id	description
1	SAS League
16	World cup
19	Champions League 06/07

Table 3.10: Raw sample of the `league` table.

3.2.6 The `club` Table

The `club` table has a row for each club and national team in the dataset. A sample of the `club` table is shown in Table 3.11.

id	name	shirt_color_1	shirt_color_2
2	FCK	220	16777215
4	Esbjerg	16777215	6724095
7	FCN	0	16763955
9	AaB	16777215	16711680
10	FCM	16777215	0
11	OB	16777215	6724095
16	Silkeborg	16777215	16724787
19	Horsens	0	16776960

Table 3.11: Raw sample of the `club` table.

id is the primary key of the club referred in `contract` and `match`. name is the name of the club.

shirt_color_1 and shirt_color_2 are the primary and secondary colour of the clubs shirts, stored as 24 bit RGB colour codes for use in the analysis interface.

3.2.7 The `lineup` and `position` Tables

The `lineup` table contains each player who participated in a given match, whether he started in the match or as a substitute, and the starting position of the players on the pitch. A sample of the `lineup` table is shown in Table 3.12.

The match_id corresponds to a match in the `match` table, see Section 3.2.4. contract_id is the contract id of the player in question, see Section 3.2.9.

substitute is a string which is "yes" if the player starts on the bench and "no" if he starts on the pitch. position is the starting position on the pitch for the player, which is a value from the `position` table. If the player starts

match_id	contract_id	position	substitute
615	240	6	yes
615	429	10	no
615	562	6	no
615	569	2	no
615	571	11	no
615	659	3	no
615	713	5	no

Table 3.12: Raw sample of the `lineup` table.

on the pitch, the position is the one the player is set to play by the analyst, otherwise the value is the standard position of the player as set in the `player` table, see Section 3.2.8.

For a sample of some of the positions, see Table 3.13. In the `position` table the primary key, id, is referred in `lineup` as position. pos is a short-hand notation for the position, and description is the full description of the position.

id	pos	description
2	DL	Left back
3	DCL	Central defender left
5	CCR	Center defender right
6	DR	Right back
10	ML	Left midfielder
11	MCL	Central midfielder left

Table 3.13: Raw sample of the `position` table.

3.2.8 The `player` Table

The `player` table represents single individual players. A sample of the `player` table is shown in Table 3.14.

id is the primary key and is used in `contract` to connect the player with his contracts. name is the full name of the player. alias is the internal alias, a shorter name, used in the analysis. spos is the standard position, the player plays on the pitch which corresponds to a description in the `positions` table, see Section 3.2.7 on the preceding page.

id	name	alias	spos
130	Dennis Flinta	Dennis Flinta	14
207	Michael Hansen	Michael Hansen	12
237	Thomas Andreasen	T. Andreasen	13
473	Morten Karlsen	Karlsen	11
601	Heath Pearce	Pearce	2
933	Bjarni Olafur Eiriksson	Olafur Eiriksson	2
934	Hördur Sveinsson	Hördur	20

Table 3.14: Raw sample of the `player` table.

3.2.9 The `contract` Table

A single player may play for multiple clubs and a national team during his career. In order to handle this, the actions in the `action` table are linked with contracts instead of players. Each contract is linked with a player, such that the same player may have multiple contracts. A sample of the `contract` table is shown in Table 3.15.

id	player_id	club_id	shirt_number	active
63	130	1	32	-1
170	237	7	14	0
414	473	7	6	0
551	130	16	18	0
569	601	7	12	0
951	207	16	8	0
952	933	16	6	0
953	934	16	27	0

Table 3.15: Raw sample of the `contract` table.

id is the primary key, and the value used in the `action` table to link a contract with an action. player_id is the player linked with the contract and club_id is the id of the club of that contract. shirt_number is the shirt number the player has now or had when he played in the club. Note that two players from the same club may have the same shirt number if only one of them is currently playing for the club.

active is a value indicating whether the contract is active or not. The values are:

-1: The contract is not active anymore.

0: The contract is a club contract and is active.

1: The contract is a national team contract and is active.

Note that a player can have any number of contracts which are not active but only a single active club contract and a single national team contract.

3.3 Specification of Match Result Data

This section presents the structure of the match result data obtained from Statman. The data is much simpler than the detailed match data but contains data for more seasons and leagues.

A diagram of the database is shown in Figure 3.3. The rest of this section presents the different tables and fields.

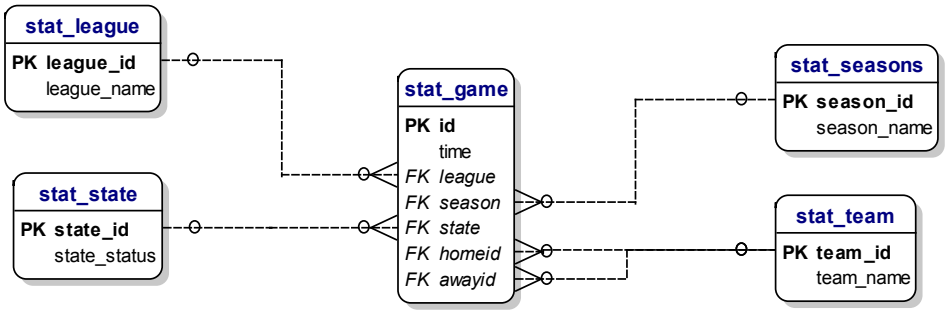

Figure 3.3: The match result database structure.

3.3.1 The `stat_game` Table

The `stat_game` is the main table, as it holds all the results from all seasons since the 1999/2000 season. The database holds 158,823 matches in all.

id denotes the id of the match, time holds the date and time of the match. state shows the status of the match which is explained in more detail in Section 3.3.4. league and season denote the league and season respectively, these are discussed further in Section 3.3.2 and Section 3.3.3 respectively. result stores the result, and lastly homeid and awayid are the ids of the home and away team, these are described further in Section 3.3.5. The matches can be matched with the detailed action data, via matching the enet_id from Table 3.7 on page 24 with id from Table 3.16. An example of the `stat_game` is shown in Table 3.16.

id	time	state	league	season	result	homeid	awayid
11581	2005-07-30 17:00:00	7	1	34	1-1	218	227
11579	2005-07-31 15:00:00	7	1	34	2-3	3	15
11582	2005-07-31 15:00:00	7	1	34	3-2	223	233
11583	2005-07-31 15:00:00	7	1	34	1-2	230	1

Table 3.16: Raw sample of `stat_game` table.

3.3.2 The `stat_league` Table

The `stat_league` table, shown in Table 3.17, holds the different leagues and their ids. league_id holds the ids and league_name holds the names of the different leagues. In this project, only the SAS league and the 1. division are used, so that matches in the `stat_game` table where league equals 1 or 2 are considered.

league_id	league_name
1	SAS League
2	1. Division
3	2. Division

Table 3.17: Raw sample of `stat_league` table.

3.3.3 The `stat_seasons` Table

The `stat_seasons` table holds the different seasons which are represented in the database. season_id holds the ids and season_name holds the textual name of the season. All seasons are used in this project. An example of the `stat_seasons` is shown in Table 3.18.

season_id	season_name
1	2000/2001
2	2001/2002
3	1999/2000

Table 3.18: Raw sample of `stat_seasons` table.

3.3.4 The `stat_state` Table

The `stat_state` table holds the possible values of the status of the matches, a match can e.g. be in the first half or finished. The state_id column holds the ids of the different statuses and state_status holds the textual version of that status. Only matches with a status of seven is used, implying that only finished matches are used. An example of the `stat_status` is shown in Table 3.19.

state_id	state_status
1	Not started
2	1. Half
3	2. Half
7	Finished

Table 3.19: Raw sample of `stat_status` table.

3.3.5 The `stat_team` Table

The `stat_team` table holds the teams together with their ids. team_id holds the team id and team_name holds the textual name of the team. There are 293 different teams in the database. An example of the `stat_team` table is shown in Table 3.20.

team_id	team_name
1	FC København
2	OB
3	AGF
15	Viborg

Table 3.20: Raw sample of `stat_team` table.

4,364 matches are in the database because only matches from the two top Danish leagues, the SAS league and the 1. division, are used.

3.4 Available Odds

To validate the model, some odds from bookmakers are needed. NordicBet has provided three sets of odds. These are their starting odds, their closing odds, and a set of average closing odds from around 150 bookmakers. Only NordicBet's closing odds and the average closing odds are used as NordicBet's starting odds are incomplete.

NordicBet is a bookmaker company working out of Isle of Man. It is owned by a company called BetPoint which also owns another bookmaker, TrioBet. The difference between TrioBet and NordicBet is that NordicBet focuses on the Scandinavian market whereas TrioBet focusses on the Baltic market.

The set of average odds is collected by a company called BetRadar, which is a company that assists bookmakers in setting their odds. Their customers are provided with a program, also called BetRadar, which collects odds from all cooperating bookmakers such that an overview of the odds market is available. Hence the bookmakers know which odds are available at the moment, such that if a set of odds gives the gamblers the possibility of a arbitrage bet, this can be corrected quickly.

The odds are delivered in a raw comma separated file, so they have to be processed in order to fit with the other data. First, the comma separated file is parsed into a data structure, then each set of odds is paired with an existing match in the database based on team names and the starting time of the match. After that, the probabilities are extracted from the odds, as described in Section 2.1 on page 5, and finally the odds are added to a table in the database with both the odds, the probabilities, and the corresponding match id.

Chapter 4

Basic Concepts

This chapter defines some general statistical terms which are used in the remainder of the report.

Section 4.1 defines expected value and expected loss followed by definitions of arithmetic mean, variance and standard deviation. Then the Poisson distribution which is a discrete probability distribution, is defined in Section 4.3 on page 35. Finally, distance measures are defined.

The probability assessors that will be created later on can be considered hypotheses, hence both hypotheses in general and test of them are reviewed. This section also defines how statistical significance is tested, reviewing both the t-test and the Wilcoxon Signed-Rank Test.

4.1 Expected Value and Expected Loss

The expected value of an instance is the sum of the probabilities of each possible outcome multiplied by its payoff or value. Other terms for expected value include expectation or mean. This implies that the expected value is the value which on average can be expected from a given scenario, if the scenario is repeated a number of times. This section is based on [DeG89].

Definition 4.1 (Expected value)**.** *The expected value for discrete outcomes is*

$$E_{\vec{p}}[\vec{x}] = \sum_i p_i x_i$$

$\vec{p}$ is the vector of probability assessments and p_i is the ith entry in that vector. $\vec{x}$ is the vector of the scores which will be given for each possibility, such that x_i, the ith entry in that vector, is the score achieved if outcome i occurs.

If a french roulette is spun a number of times with a ball and one unit is betted on red, the expected value of the one unit bet is

$$\frac{18}{37} \cdot 2 + \frac{19}{37} \cdot 0 = 0.974$$

A related term is expected loss, which intuitively can be described as what is expected to be lost in a specific scenario.

Definition 4.2 (Expected loss)**.** *The expected loss for a one unit stake on a discrete outcome is*

$$L_{\vec{p}}[\vec{x}] = 1 - \sum_i p_i x_i$$

where $\vec{p}$ and $\vec{x}$ are defined as in Definition 4.1.

4.2 Mean, Variance, and Standard Deviation

The arithmetic mean is a special case of expected value, where all scores are equally likely to happen. This implies that the probability for any of them is $\frac{1}{|\vec{y}|}$. This section is based on [DeG89].

Definition 4.3 (Arithmetic mean)**.** *The arithmetic mean is*

$$E[\vec{y}] = \frac{1}{|\vec{y}|} \sum_i^{|\vec{y}|} y_i$$

where $\vec{y}$ is a vector consisting of different continuous values of the same feature.

The variance can be described as a measure of how much the possible values are spread or dispersed around the mean value. The variance of the distribution of $\vec{y}$ is often denoted σ^2.

Definition 4.4 (Variance)**.** *The variance is*

$$Var(\vec{y}) = \sigma_{\vec{y}}^2 = E[(\vec{y} - \mu)^2] \tag{4.1}$$

where $\mu = E[\vec{y}]$ and $\vec{y}$ is a vector consisting of different continuous values of the same feature.

From this, it can be seen that the variance is never negative and a high variance implies that the different instances of the feature have a high spread around the mean value.

The standard deviation is defined as $\sqrt{\sigma_{\vec{y}}^2}$. Therefore, the standard deviation is often denoted σ. Another way of describing the standard deviation is as the deviation of the values from their arithmetic mean.

4.3 Poisson Distribution

The Poisson distribution is a discrete probability distribution commonly used in statistics and probability theory. This section is based on [DeG89].

Definition 4.5 (Poisson distribution)**.** *Let $\lambda \in \mathbb{R}$ and $\lambda > 0$ be the mean of the Poisson distribution. Then the probability for x is*

$$f(x|\lambda) = \begin{cases} \frac{e^{-\lambda}\lambda^x}{x!} & \text{for } x \in \mathbb{N} \text{ and } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that $f(x|\lambda) \geq 0$ because $e^{-\lambda} > 0$, $\lambda^x \geq 0$ and $x! \geq 1$ given the constraints.

An important property of a discrete probability distribution is that the combined probabilities sum to one. To show that the Poisson distribution is indeed a discrete probability distribution the following statement must hold:

$$\sum_{x=0}^{\infty} f(x|\lambda) = 1$$

Recall from calculus that if $\lambda \in \mathbb{R}$ then

$$e^{\lambda} = \sum_{x=0}^{\infty} \frac{\lambda^x}{x!}$$

It is now possible to prove that the Poisson distribution is a discrete probability distribution:

$$\sum_{x=0}^{\infty} f(x|\lambda) = \sum_{x=0}^{\infty} \frac{e^{-\lambda}\lambda^x}{x!} = e^{-\lambda} \sum_{x=0}^{\infty} \frac{\lambda^x}{x!} = e^{-\lambda}e^{\lambda} = e^{-\lambda+\lambda} = e^0 = 1 \quad (4.2)$$

To show that the mean, as claimed in Definition 4.5, is λ the following statement must hold:

$$E_{f(x|\lambda)}[X] = \sum_{x=0}^{\infty} x f(x|\lambda) = \lambda$$

The term where $x = 0$ yields 0 which is used next:

$$\begin{aligned} E_{f(x|\lambda)}[X] &= \sum_{x=0}^{\infty} x f(x|\lambda) = \sum_{x=1}^{\infty} x f(x|\lambda) \\ &= \sum_{x=1}^{\infty} x \frac{e^{-\lambda}\lambda^x}{x!} = \sum_{x=1}^{\infty} \frac{x e^{-\lambda}\lambda^x}{x!} = \lambda \sum_{x=1}^{\infty} \frac{e^{-\lambda}\lambda^{(x-1)}}{(x-1)!} \end{aligned}$$

Replace $y = x - 1$ and recall from (4.2) that $\sum_{x=0}^{\infty} \frac{e^{-\lambda}\lambda^x}{x!} = 1$. It is now possible to show $E_{f(x|\lambda)}[X] = \lambda$:

$$E_{f(x|\lambda)}[X] = \lambda \sum_{x=1}^{\infty} \frac{e^{-\lambda}\lambda^{(x-1)}}{(x-1)!} = \lambda \sum_{y=0}^{\infty} \frac{e^{-\lambda}\lambda^{y}}{y!} = \lambda \tag{4.3}$$

Hence $E_{f(x|\lambda)}[X] = \lambda$ as claimed.

4.4 Distance Measures

In order to determine how identical two instances are, a distance measure is needed. There are several different measures but they all share the property of being a function $d(\vec{x}, \vec{y})$, where $\vec{x}$ and $\vec{y}$ are instance vectors. This section is only concerned with distance measures for continuous features and is based on [TSK97] and [HMS01].

The measures which are examined are all metrics which implies that they all have the positivity, symmetry, and triangle inequality properties.

- $d(\vec{x}, \vec{y}) \geq 0$ for all $\vec{x}$ and $\vec{y}$ and $d(\vec{x}, \vec{y}) = 0$ if and only if $\vec{x} = \vec{y}$
- $d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x})$ for all $\vec{x}$ and $\vec{y}$
- $d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z})$ for all $\vec{x}, \vec{y}$ and $\vec{z}$

The first property, positivity, assures that the lowest score occurs when $\vec{x} = \vec{y}$ and that the score will be zero in that case. The second property, symmetry, assures that the order of the two instances does not matter. The third property, the triangle inequality, assures that if the distance between two instances via a third instance is known, then an upper bound for the distance between those two instances is known.

Distance measures build on the assumption that the different features have some degree of commensurability, implying that they have some kind of common measure. An example of this could be if all features were numeric counts of the number of passes.

The measure makes less sense if the features have different scales, like for instance number of red cards and passes. Then the number of passes will dominate the number of red cards because the values are much larger. This problem can be reduced if the features are normalised which is described in Section 5.3 on page 51.

The Euclidean distance between two instances is

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.4}$$

where $\vec{x}$ and $\vec{y}$ are two vectors of the same length.

A potential problem concerning the Euclidean distance is that since each feature contributes individually to the distance between two instances, dependent features will dominate the other features.

4.5 Test of Hypotheses

A hypothesis is an idea which is not known to be either false or correct, this implies that the hypothesis needs to be verified by a hypothesis test in order to be either rejected or accepted. This section outlines how hypotheses are set up and when a result is statistically significant.

4.5.1 Null Hypothesis

The null hypothesis, denoted H_0, is a hypothesis about population parameters, i.e. two different means of a feature. On the basis of a sample from the population, two different actions can be taken. The null hypothesis can be rejected or accepted. In some cases the desired precision for accepting or rejecting cannot be obtained with the available data. In that case, if possible, more tests cases should be made and a new test should be set up.

Often an alternative hypothesis, H_1, will be the desired result. Then the target is to reject H_0 and in the process accept H_1.

In this context, our null hypothesis could be *Bookmaker B performs on average at least as good as probability assessor a measured by some common measure* and the alternative hypothesis H_1 would be *Bookmaker B performs on average no better than probability assessor a measured by the same commons measure*. The task would then be to reject this hypothesis which would mean that probability assessor a is at least as good as Bookmaker B measured by the common measure.

4.5.2 Type 1 and 2 Errors

It is important to distinguish the different kinds of errors that can be made when accepting or rejecting a hypothesis, because some errors can be more harmful in some scenarios than others. E.g. if an automatic test is made to find out whether something is wrong with a product, it will most likely be better to detect too many errors, than too few and then make a manual test afterwards on the products in question.

In general there are two kinds of errors that can be made:

- A type 1 error is rejecting the null hypothesis even though it is true.

- A type 2 error is failing to reject the null hypothesis even though it is false.

It is easy to make a test that has no type 1 errors by always accepting the null hypothesis, but not very useful because the type 2 error then would be the highest possible. It is also possible to have no type 2 errors by always rejecting the null hypothesis but then the type 1 error would be the highest possible. A good test must have a trade-off between few type 1 and few type 2 errors, in order to be useful.

In this project the task is to make a model that performs at least as good as the current bookmakers. In order to convince a bookmaker that he should take another approach to assess odds, it would be important to show that there is a very small risk that the new method is worse, hence type 1 errors should be reduced.

4.5.3 Statistical Significance

If something is statistically significant, it informally implies that with a high probability it did not happen by chance. More formally, an acceptable significance level, α, should be decided before a test is performed. Common levels are 5%, 1%, or 0.1%, implying that there is a 0.05, 0.01, or 0.001 probability that the test will commit a type 1 error by falsely rejecting the null hypothesis. If 20 null hypotheses are rejected at the 0.05 level in 20 independent tests, one of the hypotheses is, on average, true.

When a hypothesis test is performed, a p-value is obtained. The p-value indicates what the probability is that another test will yield a different result, e.g. that the null hypothesis will be accepted instead of rejected. If the p-value is lower than the α-value, the conclusion of the test is said to be statistically significant.

Two different test methods are presented here: the t-test which is a test based on the assumption of normal distributed data and the Wilcoxon Signed-Rank Test which does not have the same assumption and hence is more versatile.

4.5.4 Dependent t-test

The t-test is useful for testing whether two datasets are sufficiently different for some hypothesis to hold. In this project, the t-test can be used to decide whether one probability assessor is better than another. The t-test assumes that the sample is normal distributed which can be tested by the Shapiro-Wilk test, presented in [SW65].

The t-test can both be performed on unpaired and paired datasets. For two datasets to be paired, they must be the same size and each item in either dataset must have exactly one partner item in the other dataset. In this project, a obvious way to pair up the items would be to pair each assessed match made by the first probability assessor or bookmaker with the same match assessed by the second probability assessor. This entails that the data has a logical dependency, implying that the dependent t-test is considered.

Let $\vec{A} = (x_{A1}, \ldots, x_{An})$ be the sampled elements from the first dataset and $\vec{B} = (x_{B1}, \ldots, x_{Bn})$ be the sampled elements from the second dataset. $\vec{D} = (d_1 = x_{A1} - x_{B1}, \ldots, d_n = x_{An} - x_{Bn})$ is the differences between the pairs. $E[\vec{D}]$ is the average difference between the pairs and is calculated as

$$E[\vec{D}] = \frac{1}{n}\sum_{i=1}^{n}(x_{Ai} - x_{Bi}).$$

Similarly $\sigma_{\vec{D}}$ is the standard deviation between the pairs which can be calculated as

$$\begin{aligned}\sigma_{\vec{D}} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((x_{A,i} - x_{B,i}) - E[\vec{D}]\right)^2} \\ &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((x_{A,i} - x_{B,i}) - \frac{1}{n}\sum_{j=1}^{n}(x_{A,j} - x_{B,j})\right)^2}\end{aligned}$$

Then the paired t-test is as stated in (4.5).

$$\begin{aligned}t =& \frac{E[\vec{D}] \cdot \sqrt{n}}{\sigma_D} \\ =& \frac{\frac{1}{n}\left(\sum_{i=1}^{n}(x_{A,i} - x_{B,i})\right)\sqrt{n}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((x_{A,i} - x_{B,i}) - \frac{1}{n}\sum_{j=1}^{n}(x_{A,j} - x_{B,j})\right)^2}} \\ =& \frac{\sum_{i=1}^{n}(x_{A,i} - x_{B,i})}{\sqrt{\sum_{i=1}^{n}\left((x_{A,i} - x_{B,i}) - \frac{1}{n}\sum_{j=1}^{n}(x_{A,j} - x_{B,j})\right)^2}}\end{aligned} \tag{4.5}$$

where t is the test statistic. Note that it is a requirement that the differences between the pairs are normal distributed.

When a t-value is obtained, it has to be compared to the level of statistical significance chosen. This can be done by an approximate method or by a table e.g. Table B.6 in [Kee95]. If the p-value in the table is above the α value, the null hypothesis cannot be rejected, hence the test is void. If the p-value from the table is below the chosen α value, the null hypothesis is rejected, in favour of an alternative hypothesis, which typically states that two elements do differ.

4.5.5 The Wilcoxon Signed-Rank Test

The Wilcoxon Signed-Rank Test [Wil45, DeG89] is a non-parametric statistical test. A non-parametric test does not require a certain distribution as opposed to a parametric test, like the t-test which requires normal distributed data. The weaker requirements make the test useful when the distribution is unknown or when the data is not distributed according to any distribution. It comes, however, at a cost since less assumptions can be made when the test is designed.

Assume that two samples, A and B, of equal size are available where each item in either sample has exactly one corresponding item in the other sample. The corresponding items forms pairs, and the total number of pairs, n, is equal to the number of items in A, which is the same number as the number of items in B. The pairs need to be independent in order for the test to work; this is not completely true in the case of football matches but it is assumed to be true so that the test can be used. Let $\vec{A} = (x_{A1}, \ldots, x_{An})$ be the sampled elements from dataset A and $\vec{B} = (x_{B1}, \ldots, x_{Bn})$ be the sampled elements from dataset B. $\vec{D} = (d_1 = x_{A1} - x_{B1}, \ldots, d_n = x_{An} - x_{Bn})$ is the differences between the pairs.

Let θ be the median so that it is the point where there are equally many pairs before and after it. Then the Wilcoxon Signed-Rank Test has the following hypotheses:

- $H_0 : \theta \leq 0$
- $H_1 : \theta > 0$

If H_0 is rejected and H_1 is accepted, θ is above 0 so sample A must have the higher median. In this case that would imply that the probability assessor used to make the probability assessments in sample A is better than the probability assessor used to make the probability assessments in sample B.

The idea is to rank all differences according to their absolute value, such that the smallest absolute value gets rank 1 and the highest gets rank n. If two or more differences have the same absolute value they all will get the average of the ranks they span. E.g. if 0.5, -0.5 and 0.5 are the three smallest values in the sample, so they should share rank 1, 2, and 3, they all would get rank $\frac{1+3}{2} = 2$ because 1 is the lowest rank and 3 is the highest rank. All rankings for positive differences are summed together forming the test statistic and this value is used to decide whether H_0 can be rejected.

To translate the sum into a probability, the mean and standard deviation of the average rank sum must be considered. It is assumed that the pairs are ordered by absolute size, so that index 1 refers to the smallest absolute

difference and index n the largest absolute difference. Let $W_i = 1$ if $d_i > 0$ and $W_i = 0$ otherwise. Then the sum of the ranks, S_n, can be expressed as:

$$S_n = \sum_{i=1}^{n} iW_i$$

Assume that the median, θ, is 0. Since the differences on average will be equally likely to be either above or below 0 (ignoring cases where $d_i = 0$), it is clear that

$$P(W_i = 0) = P(W_i = 1) = \frac{1}{2}$$

Hence the mean value $E[W_i] = \frac{1}{2}$ and the variance $Var(W_i) = \frac{1}{4}$. Recall that the pairs are independent, hence $W_1, \ldots, W_n$ are also independent. It is now possible to calculate the mean of S_n if the median, θ, is 0:

$$E[S_n] = \sum_{i=1}^{n} iE[W_i] = \sum_{i=1}^{n} i\frac{1}{2} = \frac{1}{2}\sum_{i=1}^{n} i \tag{4.6}$$

In the same way it is possible to calculate the variance:

$$Var(S_n) = Var\left(\sum_{i=1}^{n} iW_i\right) = \sum_{i=1}^{n} i^2 Var(W_i) = \sum_{i=1}^{n} i^2\frac{1}{4} = \frac{1}{4}\sum_{i=1}^{n} i^2 \tag{4.7}$$

Note that the two sums can be rewritten to

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \quad \text{and} \quad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

This implies that

$$E[S_n] = \frac{n(n+1)}{4} \quad \text{and} \quad Var(S_n) = \frac{n(n+1)(2n+1)}{24}$$

It is now possible to calculate Z_n which is S_n fitted to a standard normal distribution:

$$Z_n = \frac{S_n - E[S_n]}{\sqrt{Var(S_n)}} \tag{4.8}$$

According to [DeG89], the distribution of Z_n will converge to a normal distribution if $n \to \infty$ and $\theta = 0$. Z_n is the point in the normal standard distribution which indicates how likely it is that the median of the population of the differences is 0. In other words, if Z_n is much lower than zero it is very likely that the median is below zero and if Z_n is very high the median is probably above zero. The cumulative standard normal distribution can be used to calculate the exact probability, e.g. if $Z_n = 0$, implying that the median of the sample is 0, then the cumulative standard normal distribution will yield 0.5 probability of either of the sample having the highest values.

Algorithm 4.1 The Wilcoxon Signed-Rank Test.

1: **function** WILCOXONTEST(D)
2: $D \leftarrow$ SORTINCREASINGLY(D) ▷ Assure sorted in increasing order
3: $i \leftarrow 0$ ▷ Keep track of element currently being examined
4: $rankSum \leftarrow 0$ ▷ Keeps track of the positive ranks sum
5: **while** $i < |D|$ **do** ▷ Run until all elements are checked
6: $additional \leftarrow i + 1$ ▷ Keeps track of absolute equal values
7: **while** $|D[i]| = |D[additional]|$ **do** ▷ Next value absolute equal?
8: $additional \leftarrow additional + 1$ ▷ Try the next value
9: **end while**
10: $rank \leftarrow \frac{i+additional-1}{2}$ ▷ Average rank for these values
11: **while** $i < additional$ **do** ▷ For all absolute equal values
12: **if** $D[i] > 0$ **then** ▷ If the value is positive
13: $rankSum \leftarrow rankSum + rank$ ▷ Sum with other ranks
14: **end if**
15: $i \leftarrow i + 1$
16: **end while**
17: **end while**

18: $n \leftarrow |D|$
19: $\mu \leftarrow \frac{n(n+1)}{4}$ ▷ Calculate the expected mean
20: $\sigma \leftarrow \sqrt{\frac{n(n+1)(2n+1)}{24}}$ ▷ Calculate the expected standard deviation
21: $Z_n \leftarrow \frac{rankSum-\mu}{\sigma}$ ▷ Translate to the standard normal distribution
22: **return** $1-$ NORMALCDF(Z_n) ▷ Find the tail area of the distribution
23: **end function**

Let D be a 1-indexed array of the differences for all pairs. Then the probability that H_0 should be rejected is as stated in Algorithm 4.1.

The *rank* in line 10 is calculated that way because *additional* is 1 higher than the index of the last absolute equal value.

Lines 18-22 is the part which translates the rank sum into a useful probability based on the calculations prior stated. `NORMALCDF`(Z) is a function which returns the area under the standard normal distribution up to Z.

Chapter 5

Data Preparation

This chapter presents an overview over how the data is prepared before they are used for learning.

First the distance measure is presented which can be used for determining similarity of instances. After that the process of extracting the features from the database, how features are combined, how new features are made, and how to determine the most useful features are presented.

Lastly, the chapter presents methods for normalising the features and for selecting a learning and a test set.

5.1 Creating the Features

This section first introduces how the features can be extracted from the database. Then the concept of sliding windows, and its usage, is presented. Lastly, manipulation of the existing features to possibly filter out noise is presented.

5.1.1 Aggregating Actions into Categories Counts

A football match is very complex in the sense that play situations differ greatly. The play situations are registered as actions in the database which implies that some quantification is already made by Statman. Another quantification is made by using categories instead of action types since the categories are more general and hence occur more often. These steps are taken in order to make two matches comparable. If something only appears a few times in the database, it is not possible to tell whether it has a real impact on the outcome or not. The same is true for players, where the teams are

used instead of the players themselves. Hence, a match is represented by summing all actions by each player together at team level and then building the category counts from those actions. This implies that a match could be represented like for instance 35 crosses by the home team, 2 goals by the home team, 27 crosses by the away team, and 1 goal by the away team, and so on.

All actions in the `action` table, described in Section 3.2.1 on page 18, are single events. It is important to know which team made the actions, so the query for counts needs to be constructed such that the club of either the first or second player of the action is a grouping criteria. Since only categories and not action types are used, the query must also extract on category level. The following query counts based on the first player:

```
SELECT category_id, description, club_id, name,
  COUNT(action.id) AS count
FROM action, category, category_text, contract, club
WHERE type = action_id AND category_id = category_text.id
  AND match_id IN (610) AND contract1_id = contract.id
  AND club_id = club.id
GROUP BY club_id, category_id
ORDER BY club_id, category_id
```

Listing 5.1: The category counting query for specific teams in a match.

This query generates a result set with the category id, the category text, the club id of the players who contributed, the name of that club, and the number of events for that category in match 610. This query could easily be altered so that only players from a given club are returned, by adding `AND club_id = 2` to the `WHERE` clause. A sample from the query is shown in Table 5.1.

category_id	description	count	club_id	name
1	Finished on target	2	2	FCK
1	Finished on target	4	11	OB
2	Finished off target	15	2	FCK
2	Finished off target	4	11	OB
3	Blocked attempt	4	2	FCK
3	Blocked attempt	4	11	OB

Table 5.1: Categories summed by type and club from match 610.

5.1.2 Sliding Windows

All predictions on what happens in a match must be based on what happened in previous matches and possibly other information surrounding the team

prior to the match.

According to Kresten Buch, the way a team performs depends on their form [sta]. The form of the team is reflected in the different actions which the team made in their last matches. Hence in order to predict the performance of a team in a match, the actions of that team in the last few matches can be used. *window* is used to denote the last n matches. Note that the matches must be in the same season because two teams are relegated from the SAS league and two teams are promoted into it and because the players in the squad of a team most likely change significantly in the summer break. The result of this is that there is no data available in the first *window* rounds of the season.

The strength of a team is based on both its ability to create and utilise chances but also its ability to keep their opponents from doing the same. Hence, actions made by a team and against it should both be considered. This implies that for the probability assessment of a match, there are four different values for each category; category counts for the home team in the last *window* matches, category counts against the home team in the last *window* matches, category counts for the away team in the last *window* matches, and category counts against the away team in the last *window* matches.

An example of the process, with a window size of four, is presented in Tables 5.2, 5.3, and 5.4.

Date	Home	Away	Crosses (h)	Crosses (a)
28/8 05	Horsens	AaB	39	**38**
11/9 05	AaB	FC Nordsjælland	**34**	40
17/9 05	AGF	AaB	32	**45**
21/9 05	AaB	Sønderjyske	**41**	29

Table 5.2: Number of crosses in four succeeding matches with AaB.

Date	Home	Away	Crosses (h)	Crosses (a)
27/8 05	Viborg	OB	**29**	41
11/9 05	FCM	Viborg	39	**34**
21/9 05	Viborg	Brøndby	**42**	41
25/9 05	Esbjerg	Viborg	37	**38**

Table 5.3: Number of crosses in four succeeding matches with Viborg.

Table 5.4 is made by summing the category counts for and against each team in the last four matches and is the instance representing that match. h/f means the count for the home team, h/a the count against the home

Home	Away	Cros. (h/f)	Cros. (h/a)	Cros. (a/f)	Cros. (a/a)
Viborg	AaB	143	158	158	140

Table 5.4: The summed crosses from Table 5.2 on the preceding page and Table 5.3 on the facing page for Viborg-AaB.

team, a/f the count for the away team, and a/a the count against the away team. For instance, the 143 crosses for Viborg in the last four matches, all marked in boldface font, are calculated by summing the 29 crosses Viborg made against OB, the 34 against FCM, the 42 against Brøndby, and the 38 against Esbjerg. Note that the construction of the instance implies that there are four features for each category type so far.

A query for the last *window* matches is needed for the extraction. An example for the last four matches is presented in Listing 5.2.

```
SELECT id FROM match
WHERE (home_id = 2 OR away_id = 2) AND status = 5
  AND date < 1155333600
ORDER BY date DESC LIMIT 0, 4
```

Listing 5.2: The match selection query.

This query first selects all matches where the home or away team has id 2, by the statement `home_id = 2 OR away_id = 2`. Afterwards, it is assured that the analysis of the match is finished, implying that `status = 5`. In order to get only the matches before a specific date, the time stamp for all matches must be below 1155333600 which implies that they are played before the 12th August 2006. The available matches are ordered decreasing by `date` and the first four matches are selected. The result of the query, the last four matches, is 587, 575, 573 and 540.

It is possible to combine the match selection query from Listing 5.2 with the category counting query from Listings 5.1 on page 45, such that the counting is performed on previous matches. This is done by replacing 610 in the `WHERE` clause of the category counting queries with the SQL query for selecting matches.

5.1.3 Manipulation of Categories

Some actions are registered with coordinates and all of them are registered with time stamps. Moreover, the line-up is also included in the database. From these properties it is possible to construct additional categories.

Six categories are made from the "Finished" categories by filtering them according to their coordinates:

- "Finished off target in the goal area"
- "Finished off target in the penalty area except the goal area"
- "Finished off target outside the penalty area"
- "Finished on target in the goal area"
- "Finished on target in the penalty area except the goal area"
- "Finished on target outside the penalty area"

One reason, why filtered categories could perform better than the original categories, is that the filter process could clean the categories for noise. In the case of "Finished", the noise could be finishes made from odd angles or far from the goal.

The information of line-ups from the previous matches is also used as a custom category: "Players missing from last match". All other categories are constructed based on data from previous matches which is not necessary in this case because the line-ups are known in advance. Hence, it can be assumed that something known about the match gives additional information about the match, compared to something that happened in previous matches.

5.2 Feature Reduction

The dataset consists of a large number of overlapping features, implying that the same information is represented more than once, e.g. some dependency between the number of goals and finishes on goal in a match can be expected. Another possibility is that they are unrelated to the class variable, implying e.g. that it is plausible that the number of throw ins that the home team has in the middle of the field is unrelated to the outcome of the match. Hence, it is often possible to find a feature set which contains the same amount of information but has a lower number of features.

Furthermore since the dataset has a large number of features, it has a high dimensionality, and this affords a number of difficulties. Datasets, which have a high dimensionality, have a higher probability of containing irrelevant and redundant features. A further problem is the curse of dimensionality which refers to the phenomenon that data analysis becomes significantly harder as the dimensionality of the data increases [TSK97, p.51]. This is caused by the fact that when the number of features increases, the number of dimensions increases which again makes the data lie sparsely in Euclidean space. Therefore, a larger number of training instances is needed to make a qualified classification. As the number of training examples in this project is limited, the number of features needs to be reduced.

Feature reduction can basically be done in two ways: either by aggregating features or by selecting a subset of features.

5.2.1 Aggregation of Similar Features

Reducing the number of features can be accomplished by projecting data from a high dimension into a lower dimension. This implies that features are aggregated, such that the dataset contains the same amount of information but the dimensionality is lower.

There are a number of ways to aggregate features, e.g. by linear transformation. In this case, a number of action types are combined linearly, such that the result is the category types which are then used instead of the action types in the linear equation.

The categories have a higher numeric value than the summed action types because a detailed action type like "finished with right foot - goal - deflected by team mate" rarely occurs. This implies that the numeric value for "finished with right foot - goal - deflected by team mate" is mostly 0, in some cases 1, but very rarely higher. By creating categories, some of this detailed information disappears, but whether a goal was scored using the right or the left foot is potentially irrelevant.

5.2.2 Aggregation of Home and Away Features

For each category type four different values are made as described in Section 5.1.2 on page 45. Using a distance measure like the Euclidean, described in Section 4.4 on page 36, could easily provide the wrong distance, as illustrated in Table 5.5. The data in the table is assumed made using a sliding window of size four.

Instance	$Home_{for}$	$Home_{against}$	$Away_{for}$	$Away_{against}$
A	7	3	3	7
B	7	7	3	3
C	5	5	5	5

Table 5.5: Number of goals in last four matches for three instances.

Using a Euclidean distance measure, the distance between A and C is 4 as is the distance between B and C, but the two instances are very different. Consider instance C: both teams have scored and conceded equally many goals so the instances are similar. A is different from C because the home team has a goal difference of +4 while the away team has a goal difference of -4. This implies that in instance A, the home team should be a huge favourite, which differs from the scenario in C where the two teams are about equally good. In instance B, both the home and away team has a goal difference of 0, so they should be considered about equally good, as it were the case in C.

This short example shows that when the four different values are presented for each category, the Euclidean distance could be a bad measure so other strategies should be considered.

One approach is to shrink the four values into one, an overall value. The idea is that events made by the home team and against the away team count positively and that events made by the away team or against the home team counts negatively. The formula is

$$overall = home_{for} + away_{against} - (away_{for} + home_{against}) \tag{5.1}$$

It is straight forward to see that if the home team made a large number of the actions in the last few games and the away team's opponents also did that, the value will tend to be positive. On the other hand the value will be negative if the away team made a lot of them in the last matches in combination with the home team's opponents.

Applying 5.1 to the instances in Table 5.5 would yield a value of 8 for A and 0 for both B and C which would result in C being classified based on Bs class variable which were the intention.

5.2.3 Feature Subset Selection

If a large number of different features exist in a dataset, it is plausible that not all of these are necessary. Hence, a subset can be selected from the original set of features, such that the probability assessment maintains the same level of accuracy or even increases its accuracy.

The creation of a subset can be done with three different approaches: the embedded, the filter, and the wrapper. In the embedded approach, the selection of features takes place as a part of the classifier algorithm, such as in the decision tree algorithm. The filter approach takes place before the classifier algorithm is run and is independent of the algorithm. Every feature gets a value according to how much information it contains, for instance using information gain [Mit97], and the best can then be selected. The wrapper approach uses the probability assessor to test which subset is the best. The optimal way of doing this is to use the power set of features, but this is rarely possible as with n features the total number of possible subsets are $2^n - 1$ non-empty. This implies that not all possible subsets can be tested and an approach which only tests some of the possible subsets is needed. Unfortunately, no polynomial time algorithm exists which guarantee to yield the best subset [HMS01].

5.3 Normalisation of Features

The features used in data mining can have very different ranges which has a negative impact on the accuracy of data mining algorithms which use distance measures. For instance, the number of passes vary from 100 to 1,000 between games which are much higher values than the numbers of goals which vary between 0 and 10. Some algorithms produce a probability assessor where the impact of the passes is huge. To ensure that number of goals and passes have the same initial impact, the features should be normalised. This is done by applying a normalisation function, f, to each value, x_i, in the dataset to create x'_i so that $x'_i = f(x_i)$

One simple way to do that is by using the min-max normalisation:

$$x'_i = \frac{x_i - min(\vec{x})}{range(\vec{x})} = \frac{x_i - min(\vec{x})}{max(\vec{x}) - min(\vec{x})} \tag{5.2}$$

where $\vec{x}$ is the set of features and $x_i \in \vec{x}$. The min-max normalisation ensures that all values in $\vec{x}$ be normalised to the range $0 \leq x'_i \leq 1$.

Another approach is to use Z-score normalisation [Abd06]. The Z-score normalisation centres the distribution around 0 with a variance and standard deviation of 1. Variance and standard deviation are described in Chapter 4 on page 33:

$$x'_i = \frac{x_i - E[\vec{x}]}{\sigma_{\vec{x}}} \tag{5.3}$$

The Z-score normalisation usually results in all values being within the range $-4 < x'_i < 4$.

5.4 Splitting the Dataset

A probability assessor first needs to be learned. The purpose of the learning process is to enable the model to make the correct probability assessment for a given instance.

To perform the learning, a function is needed which uses a pair of vectors: one consisting of learning instances and one with the corresponding class labels. This kind of learning is called supervised learning, as opposed to unsupervised learning where no class labels are known.

The learning should work in a way such that the prediction error is minimised, implying that the model should perform as well as possible given the knowledge of the learning set. There is one consideration though, namely overfitting. Overfitting occurs when a model has been extremely well fitted to a learning set without considering what will happen when an instance,

which has not been observed before, is observed. The model will then perform badly on the test set. This is described further in Section 6.2.3 on page 66.

When a hypothesis is found, the result must be verified. Verification is done by testing the hypothesis on a test set which the hypothesis has not been trained on. This implies that the probability assessor must never have seen the test set before.

To achieve the possibility of verification, the dataset must be divided into two sets prior to the learning: the learning set is denoted L and the validation set is denoted V. The same data should not be used for both learning and testing because the model would then have learned on the same data that it uses for testing, implying that $L \cap V = \emptyset$.

One potential problem with this approach is that the entire dataset is not used for learning which does not utilise the full potential of the dataset. This is especially a problem if the dataset is very limited in size. If new data is generated from time to time, this problem is reduced since all the currently known data can be used for learning and the final evaluation can be done on the data which was generated while the tests and fittings were done.

Chapter 6

Probability Assessors

This chapter first presents scoring rules for evaluating probability assessors, including both absolute and pair wise scoring rules. Then the basics for learning probability assessors, including how the data should be split and how to reduce overfitting, are presented. Finally, the test plan is presented which includes descriptions of which learning algorithms are chosen and on what data they learn.

6.1 Scoring Rules for Probability Assessors

Scoring rules are needed for evaluation of probability assessors such that the best probability assessors can be used and the others avoided. A probability assessor must make its probability assessment prior to the observation of the outcome. After the outcome is observed, the quality of the assessments can be determined by the use of a scoring rule. A scoring rule is a function which from one or two probability assessments and an observed outcome, yields a score. There are generally two types of scoring rules: absolute scoring rules and pair wise scoring rules.

An absolute measure calculates an individual score for each probability assessment depending on the observed outcome. As each probability assessor has a single deterministic score on each instance in a given dataset, the absolute measure can be used to rank the probability assessors.

A pair wise measure calculates a score for two probability assessors in comparison with each other. Assuming that a scoring rule is applied which can determine which of the two probability assessors is the best with regard to a number of observed outcomes, it is possible to determine a winner. Though it seems trivial to determine the winner given an observed outcome, as the probability assessor with the highest probability on the observed outcome

should be regarded the best, it is more complicated over a number of instances. The pair wise scoring rules make it possible to make a partial ranking of a number of probability assessors according to the number of victories they obtain, if all possible pairs are compared.

Notation

Before the scoring rules are explained, some notation is needed. The set of all outcomes is denoted D and is considered ordered, so that it is possible to refer to the outcomes as outcome $1, \ldots, n$, where n is the number of elements in D.

Three vectors exist $\vec{p} = \langle p_1, \ldots, p_n \rangle$, $\vec{r} = \langle r_1, \ldots, r_n \rangle$, and $\vec{d} = \langle d_1, \ldots, d_n \rangle$. $\vec{p}$ represents the output of a given probability assessor, such that $\sum_i p_i = 1$ and $0 \leq p_i \leq 1$ e.g. $\langle 0.3, 0.4, 0.3 \rangle$ in a scenario with 3 potential outcomes. $\vec{r}$ represents the probability assessment which a given probability assessor provides to the scoring rule, $\vec{r}$ is not necessarily identical to $\vec{p}$. $\vec{d}$ represents the outcome vector, such that $\sum_i d_i = 1$ and $d_i \in \{0, 1\}$ e.g. $\langle 0, 1, 0 \rangle$ in a scenario with 3 potential outcomes, where the 2nd possibility is the observed outcome.

A set of vectors, h, exists in which $\vec{h}^1, \ldots, \vec{h}^n \in h$. $\vec{h}^k$ represents k vectors of length k. For all $1 \leq j \leq k$ exists a vector h^k_j where $h^k_{j,j} = 1$ and $h^k_{j,i} = 0$ for all $i \neq j$, $1 \leq i \leq k$, e.g. $\vec{h}^3_2 = \langle 0, 1, 0 \rangle$. This set of vectors is used to emulate all possible outcome vectors.

6.1.1 Absolute Scoring Rules

An absolute scoring rule is a function, S, which takes two vectors $\vec{p}$, a probability assessment from probability assessor c, and $\vec{d}$, the outcome vector, as input and yields a score for c as shown in (6.1):

$$S : (\vec{p}, \vec{d}) \rightarrow \mathbb{R} \tag{6.1}$$

The absolute scoring rules allow ranking to be performed easily by calculating the score for each instance, a probability assessor assesses and taking the average of those values.

Definition 6.1 (Expected score, absolute scoring rules)**.** *The expected score is a special case of expected value, see Section 4.1 on page 33, where $\vec{x}$ in $E_{\vec{p}}[\vec{x}]$ is calculated from the scoring rule and the assessment, $\vec{r}$, that the probability assessor uses. x_i is calculated as the score which would be given if outcome i occurs, i.e. $x_i = S(\vec{r}, \vec{h_i})$. The expected score for the absolute scoring rules is shown in (6.2).*

$$E_{\vec{p}}[S(\vec{r}, \vec{h^n})] = \sum_i p_i \cdot S(\vec{r}, \vec{h^n_i}) \tag{6.2}$$

In some cases, it is possible for a probability assessor to achieve a better expected score if it uses a specific strategy to make the probability assessment. Hence, a desired property of a scoring rule is properness. In a proper scoring rule, the probability assessor maximises its expected score by using the probability assessment which it believes to be correct, rather than one altered to fit the scoring rule. More formally Definition 6.2 must hold.

Definition 6.2 (Properness, absolute scoring rules)**.** *Let $\vec{p}$ denote the probability assessment made by the probability assessor, n denote the length of $\vec{p}$ and $\vec{h}$ defined as before. Then the absolute scoring rule S is proper if*

$$E_{\vec{p}}[S(\vec{p}, \vec{h^n})] \geq E_{\vec{p}}[S(\vec{r}, \vec{h^n})]$$

for all $\vec{r} \in [0,1]^n$ where $\sum_{i=1}^{n} r_i = 1$.

An example of a scoring rule which is not proper is a scoring rule which outputs the prediction of the outcome as the score such that

$$S(\vec{p}, \vec{d}) = \sum_i p_i \cdot d_i. \tag{6.3}$$

If a probability assessor for a given instance with two potential outcomes calculates and outputs $\vec{p} = \langle 0.7, 0.3 \rangle$, it has the expected score

$$\begin{aligned} E_{\vec{p}}[S(\vec{p}, \vec{h^2})] &= \sum_i p_i \cdot \left(\sum_j p_j \cdot h_{i,j}^2 \right) \\ &= 0.7 \cdot (0.7 \cdot 1 + 0.3 \cdot 0) + 0.3 \cdot (0.7 \cdot 0 + 0.3 \cdot 1) = 0.58 \end{aligned}$$

However, the expected score for this rule can be increased without any further knowledge. Assume that the probability assessor instead outputs $\vec{r} = \langle 1, 0 \rangle$ as its probability assessment, so that the expected score would be

$$\begin{aligned} E_{\vec{p}}[S(\vec{r}, \vec{h^n})] &= \sum_i p_i \cdot \left(\sum_j r_j \cdot h_{i,j}^2 \right) \\ &= 0.7 \cdot (1 \cdot 1 + 0 \cdot 0) + 0.3 \cdot (1 \cdot 0 + 0 \cdot 1) = 0.7 \end{aligned}$$

This example shows that some scoring rules favour probability assessors which change their probability assessment in order to achieve a higher expected score. Though this is an extreme example, it shows that a scoring rule, which is not proper, could encourage the probability assessors to output a classification rather than a probability assessment. As mentioned in Section 1.2 on page 3, the intent of this project is not to classify the actual outcome but to make good probability assessments for the matches. Hence, the properness property of a scoring rule is important in order to achieve that goal.

There are a number of different absolute scoring rules, e.g. the quadratic scoring rule, the logarithmic scoring rule, the spherical scoring rule, and the zero-one scoring rule. The quadratic scoring rule and the logarithmic scoring rule are described below, the others can be found in [WM68] and [GR04].

As an absolute scoring rule yields an absolute and deterministic score for a probability assessment on each instance in a given dataset, the score can be viewed as the quantitative measure for the strength or quality of the classifier applied to that particular dataset.

Quadratic Scoring

The quadratic scoring rule is a proper scoring rule [WM68]. It sets the score to the sum of the quadratic difference between all pairs (p_i, d_i), so that

$$Q(\vec{p}, \vec{d}) = 1 - \sum_i (p_i - d_i)^2. \tag{6.4}$$

If the jth possible outcome is observed, then $d_j = 1$ and $d_i = 0$ for all $i \neq j$. This implies that (6.4) yields (6.5).

$$Q(\vec{p}, \vec{d}) = 1 - (p_j - 1)^2 - \sum_{i \neq j} p_i^2 = 2p_j - \sum_i p_i^2 \tag{6.5}$$

Note that if the probability assessment on the outcome, p_j, is 1, so that the prediction on all the other outcomes is 0, this maximises the score. Conversely, as p_j goes towards zero, the score goes towards the sum of all probabilities squared.

$$\begin{aligned} \lim_{p_j \to 0} Q(\vec{p}, \vec{d}) &= -\sum_{i \neq j} p_i^2 \geq -1 \\ \lim_{p_j \to 1} Q(\vec{p}, \vec{d}) &= 1 \end{aligned}$$

The fact that the differences between pairs of (p_i, d_i) are squared results in the score decreasing quadratically. The quadratic scoring rule takes the entire vector of the probability assessment, $\vec{p}$, into account, when the score is calculated. Let C_1 and C_2 be two probability assessors which each makes a probability assessment on a given match. The probability assessment of C_1 is $\langle 0.3, 0.35, 0.35 \rangle$ and the probability assessment of C_2 is $\langle 0.35, 0.6, 0.05 \rangle$. Let the outcome be 1, then the quadratic scoring rule gives C_1 a score of 0.265, while C_2 gets a score of 0.215, implying that C_1 is best.

At first, this is a little surprising because C_1 has a lower probability on the outcome but it is the result of the fact that all probabilities are taken into account. This is a property which ensures that the quadratic scoring rule

penalises probability assessors which have high probabilities on outcomes which are not observed. Since the bookmaker only pays out money if the outcome actually occurs this property is not desired. Hence, the quadratic scoring rule is not used for model tuning.

Logarithmic Scoring

The logarithmic scoring rule [WM68] sets the score to the natural logarithm of the probability of the actual outcome, so that

$$L(\vec{p}, \vec{d}) = \sum_i \ln(p_i) \cdot d_i \tag{6.6}$$

where $\infty \cdot 0 = 0$. If the jth outcome is observed, then $d_j = 1$, $d_i = 0$ for all $i \neq j$ and $\sum_k d_k = 1$, and (6.6) yields

$$L(\vec{p}, \vec{d}) = \ln(p_j)$$

Note that in contrast to the quadratic scoring rule, only p_j is taken into account when calculating the score. The bounds for the quadratic score can be seen below.

$$\begin{aligned} \lim_{p_j \to 0} L(\vec{p}, \vec{d}) &= -\infty \\ \lim_{p_j \to 1} L(\vec{p}, \vec{d}) &= 0 \end{aligned}$$

Because only p_j is taken into account, a higher probability on the observed outcome yields a better score. Reconsider the example from before with two probability assessors C_1 and C_2 making probability assessments for a given match. Let the outcome be 1, then the logarithmic scoring rule gives C_1 a score of -0.523 while C_2 gets a score of -0.456, implying that C_2 is best.

From this is seen that a significant difference between the quadratic scoring rule and the logarithmic scoring rule is that the logarithmic scoring rule does not take the entire probability assessment into account when it calculates the score for the probability assessment, but only the probability on the actual observed outcome. This also implies that the logarithmic scoring rule gives a great penalty ($-\infty$) to probability assessors which set the probability of the observed outcome to 0. Actually, if the scores of all matches were averaged, the probability assessor with the 0 probability would lose or tie no matter how well it performs on the rest of the instances.

Theorem 6.3 (Logarithmic scoring rule is proper)**.** *The logarithmic scoring rule is a proper scoring rule.*

Proof. Assume that the logarithmic scoring rule is not proper. Then a better expected score is achieved by using vector $\vec{r}$ instead of $\vec{p}$.

The expected score, $E_{\vec{p}}[S(\vec{r}, \vec{h^n})]$, using the logarithmic scoring rule is

$$E_{\vec{p}}[S(\vec{r}, \vec{h^n})] = \sum_i p_i \sum_j \ln(r_i) \cdot h^n_{i,j} = \sum_i p_i \ln(r_i) \tag{6.7}$$

Since $\sum_i r_i = 1$, $E_{\vec{p}}[S(\vec{r}, \vec{h^n})]$ can be maximised using a Lagrange multiplier [Zwi03, pp.389–390]. Let $n = |\vec{p}|$, $f(r_1, \ldots, r_n) = \sum_{i=1}^n p_i \ln(r_i)$ and $g(r_1, \ldots, r_n) = \sum_{i=1}^n r_i = 1$.

For $k = 1, \ldots, n$, it is required that $\frac{d}{dr_k}(f + \lambda(g-1)) = 0$.

$$\begin{aligned}
0 &= \frac{d}{dr_k}(f + \lambda(g-1)) \\
&= \frac{d}{dr_k}\left(\sum_{i=1}^n p_i \ln(r_i) + \lambda(\sum_{i=1}^n r_i - 1)\right) \\
&= \frac{d}{dr_k}\left(\sum_{i=1}^n p_i \ln(r_i)\right) + \frac{d}{dr_k}\left(\lambda(\sum_{i=1}^n r_i - 1)\right) \\
&= p_k \frac{1}{r_k} + \lambda \\
\Leftrightarrow\ & r_k = \frac{-1}{\lambda} p_k
\end{aligned}$$

It is known that $\sum_i r_i = \sum_i p_i = 1$ and that $r_i = \frac{-1}{\lambda} p_i$, $i = 1, \ldots, n$, so λ must be -1, hence $\vec{p}$ will maximise the expected score. This shows that Theorem 6.3 holds so that the logarithmic scoring rule is a proper scoring rule. □

6.1.2 Pair Wise Scoring Rules

A pair wise scoring rule involves two probability assessors. The objective is to determine which scoring rule is the better by letting them compete. A pair wise scoring rule is a function which takes three vectors $\vec{p_1}$, $\vec{p_2}$, $\vec{d}$ as input and yields a score as shown in (6.8). $\vec{p_1}$ is the probability assessment yielded by probability assessor C_1 and $\vec{p_2}$ is the probability assessment yielded by probability assessor C_2. $\vec{d}$ is defined as in Section 6.1.

$$S : (\vec{p_1}, \vec{p_2}, \vec{d}) \to \mathbb{R} \tag{6.8}$$

A pair wise scoring rule must be antisymmetric implying that

$$S(\vec{p_1}, \vec{p_2}, \vec{d}) = -S(\vec{p_2}, \vec{p_1}, \vec{d}) \quad \text{and} \quad S(\vec{p_1}, \vec{p_1}, \vec{d}) = 0. \tag{6.9}$$

$$\tag{6.10}$$

The pair wise scoring rules are different from the absolute ones, in that they need two probability assessments in order to produce a score.

Definition 6.4 (Expected score, pair wise scoring rules)**.** *Let $\vec{p_1}$, a vector of length n, be the probability assessment calculated by probability assessor C_1, $\vec{r}$ be the probability assessment probability assessor C_1 inputs in the scoring rule, $\vec{p_2}$ be the probability assessment made by probability assessor C_2 and $\vec{h}^n$ defined as above. The expected score of $\vec{r}$ from probability assessor C_1's point of view, is*

$$E_{\vec{p_1}}[S(\vec{r}, \vec{p_2}, \vec{h^n})] = \sum_i p_{1,i} \cdot S(\vec{r}, \vec{p_2}, \vec{h_i^n}). \tag{6.11}$$

Properness is a desired property of pair wise scoring rules as it is with absolute scoring rules. There are two kinds of properness for pair wise scoring rules. Strong properness, Definition 6.5, requires the expected score by using the calculated probability assessment to be better than any alteration no matter what the opponents probability assessment is.

Definition 6.5 (Strong properness, pair wise scoring rules)**.** *Let $\vec{p_1}$ denote the probability assessment made by probability assessor C_1, $\vec{p_2}$ denote the probability assessment made by probability assessor C_2, and n denote the length of $\vec{p_1}$. Then the pair wise scoring rule S is strongly proper if*

$$E_{\vec{p_1}}[S(\vec{p_1}, \vec{p_2}, \vec{h^n})] \geq E_{\vec{p_1}}[S(\vec{r}, \vec{p_2}, \vec{h^n})]$$

for all $\vec{r}, \vec{p_2} \in [0,1]^n$ where $\sum_{i=1}^n r_i = \sum_{i=1}^n p_{2,i} = 1$.

Definition 6.6, weak properness, requires the expected score by using the calculated probability assessment to be better than any alteration under the assumption that the opponent's probability assessment is the same as the calculated probability assessment. Weak properness can be used in cases where the opponents probability assessment is unknown, in such a case the best guess of the opponents probability assessment is the calculated probability assessment.

Definition 6.6 (Weak properness, pair wise scoring rules)**.** *Let $\vec{p_1}$ denote the probability assessment made by probability assessor C_1, and n denote the length of $\vec{p_1}$. Then the pair wise scoring rule S is weakly proper if*

$$E_{\vec{p_1}}[S(\vec{p_1}, \vec{p_1}, \vec{h^n})] \geq E_{\vec{p_1}}[S(\vec{r}, \vec{p_1}, \vec{h^n})] \tag{6.12}$$

for all $\vec{r} \in [0,1]^n$ where $\sum_{i=1}^n r_i = 1$.

using (6.9), (6.12) is equivalent to

$$0 \quad \geq \quad E_{\vec{p_1}}[S(\vec{r}, \vec{p_1}, \vec{h^n})]$$

Note that it is only necessary to prove that p_1 cannot be changed for an expected gain because a pair wise scoring rule must be antisymmetric.

The scores from a pair wise scoring rule cannot directly be used to rank more than two probability assessors. This implies that if a ranking of three or more probability assessors is needed, some way of doing so is also needed. One obvious approach would be to let the probability assessors compete in a round robin fashion and then rank the probability assessors according to the number of victories they obtain against other probability assessors. This ranking would only be partial since it is plausible that two probability assessors obtain the same number of victories.

One example of a pair wise scoring rule, the Bookmaker-Gambler scoring rule, is explained below.

Bookmaker-Gambler Scoring

The Bookmaker-Gambler scoring rule is a pair wise scoring rule for the particular problem domain of this project. As this is a pair wise scoring rule, two probability assessors are needed, C_1 and C_2. Furthermore, two vectors $\vec{p_1} = \langle p_{1,1}, \ldots, p_{1,n} \rangle$ and $\vec{p_2} = \langle p_{2,1}, \ldots, p_{2,n} \rangle$ are needed. $\vec{p_1}$ represents the probability assessment given by probability assessor C_1, and $\vec{p_2}$ represents the probability assessment given by probability assessor C_2.

The scoring rule is divided into two well defined parts: the first, where C_1 is the bookmaker and c_2 is the gambler, and the second where C_2 is the bookmaker and c_1 is the gambler.

Some assumptions about betting are made: it is assumed that the gambler only bets on outcomes on which he can achieve the highest expected gain. If the same gain is expected in multiple cases, the stakes are distributed evenly. In order to achieve fairness, both probability assessors must have a pay-back-percentage of 1 when acting as bookmakers.

First, the gambler compares the odds from the bookmaker, $\vec{o}_{bookmaker}$, with the probabilities calculated by the gambler, $\vec{p}_{gambler}$, for all possible outcomes. This is calculated by simulating a one unit bet on each outcome, calculating the stake which would be returned, and multiplying that with the probability that the outcome is observed. The expected return for outcome i is named t_i.

$$t_i = (1 \cdot o_{bookmaker,i}) \cdot p_{gambler,i} = \frac{p_{gambler,i}}{p_{bookmaker,i}}$$

The rewriting is based on the proportional relationship between odds and probabilities, see Section 2.1.1 on page 5, that is $o_i = \frac{1}{p_i}$.

The task is now to find the i values which maximise t_i. The stake vector, $\vec{s}$, is now made where

$$s_i = \begin{cases} 1 \text{ if } t_i = \max_j t_j \\ 0 \text{ if } t_i < \max_j t_j \end{cases}$$

To ensure that both probability assessors gamble for the same amount of units, the vector is normalised into $\vec{s'}$, such that

$$s'_i = \frac{s_i}{\sum_j s_j}$$

Now any vector, $\vec{s'}$, satisfies $\sum_i s'_i = 1$, which implies that any gambler will bet for exactly one unit.

When the result vector, $\vec{d}$, is known, the result, R, for assessor c playing as gambler can be found:

$$\begin{aligned} & R_{c_1,gambler}(\vec{d},\vec{s'},\vec{o}_{bookmaker}) &&= \sum_i d_i \cdot s'_i \cdot o_{bookmaker,i} - 1 \\ \Leftrightarrow \quad & R_{c_1,gambler}(\vec{d},\vec{s'},\vec{p}_{bookmaker}) &&= \sum_i d_i \cdot s'_i \cdot \frac{1}{p_{bookmaker,i}} - 1 \end{aligned}$$

If the outcome is i and the gambler has a stake on it, the gambler gets the return of stake times odds. Recall that 1 is subtracted from the result because it is the stake of the gambler

It is now possible to calculate the score for both probability assessors which is done by letting them both act as a gambler and as a bookmaker. The result for the first probability assessor, C_1, is:

$$\begin{aligned} BG(\vec{p_1},\vec{p_2},\vec{h}) &= R_{c_1,gambler} - R_{c_2,gambler} \\ &= \sum_i d_i \cdot s'_{c_1,i} \cdot \frac{1}{p_{c_2,i}} - 1 - (\sum_i d_i \cdot s'_{c_2,i} \cdot \frac{1}{p_{c_1,i}} - 1) \\ &= \sum_i d_i \cdot \frac{s'_{c_1,i}}{p_{c_2,i}} - \sum_i d_i \cdot \frac{s'_{c_2,i}}{p_{c_1,i}} \\ &= \sum_i d_i \cdot \left(\frac{s'_{c_1,i}}{p_{c_2,i}} - \frac{s'_{c_2,i}}{p_{c_1,i}} \right) \end{aligned}$$

The result for the probability assessor is used as the score for the given probability assessment, the probability assessor made. It is clear that the expression is antisymmetric, so if the order of C_1 and C_2 is changed, the score will be the negative value of C_1's result.

The complete algorithm for calculating the score, using the Bookmaker-Gambler scoring rule, is given in Algorithm 6.1. It uses the function

Algorithm 6.1 The Bookmaker-Gambler scoring function.

function BG($\vec{p_1}, \vec{p_2}, \vec{d}$)
  $\vec{s'_1} \leftarrow$ CALCULATESTAKES($\vec{p_1}, \vec{p_2}$) ▷ Calculate the stakes for p_1
  $\vec{s'_2} \leftarrow$ CALCULATESTAKES($\vec{p_2}, \vec{p_1}$) ▷ Calculate the stakes for p_2

  $score \leftarrow \sum_i d_i \cdot (\frac{s'_{1,i}}{p_{2,i}} - \frac{s'_{2,i}}{p_{1,i}})$ ▷ Calculate the score
  return *score* ▷ Return the score
end function

Algorithm 6.2 The CALCULATESTAKES function.

function CALCULATESTAKES($\vec{p_1}, \vec{p_2}$)
  for $i \leftarrow 1$ **to** $|\vec{d}|$ **do**
    $t_i \leftarrow \frac{p_{1,i}}{p_{2,i}}$ ▷ Calculate the expected payback
  end for

  $max \leftarrow \max_i t_i$ ▷ Find the maximal payback
  for $i \leftarrow 1$ **to** $|\vec{d}|$ **do**
    if $t_i = max$ **then** ▷ Is the payback the largest achievable?
      $s_i \leftarrow 1$ ▷ If so, place a bet on that outcome
    else
      $s_i \leftarrow 0$ ▷ Otherwise do not
    end if
  end for

  $sum \leftarrow \sum_i s_i$ ▷ Find the sum of all stakes
  for $i \leftarrow 1$ **to** $|\vec{d}|$ **do**
    $s'_i \leftarrow \frac{s_i}{sum}$ ▷ Normalise each stake
  end for

  return $\vec{s}$ ▷ Return the stake vector
end function

CalculateStakes, given in Algorithm 6.2, as a sub function to calculate the stakes.

When $BG(\vec{p_1}, \vec{p_2}, \vec{d})$ has been computed, based on the probability assessments made by C_1 and C_2, the overall winner is C_1 if the score is positive, the result is 0 if the two probability assessors are equally good or identical, and if the result is negative C_2 is the winner.

The Bookmaker-Gambler scoring rule is not strongly proper which can be seen from the following example. Consider $\vec{p_1} = \langle 0.5, 0.3, 0.2 \rangle$, $\vec{r} = \langle 0.41, 0.3, 0.29 \rangle$ and $\vec{p_2} = \langle 0.4, 0.3, 0.3 \rangle$. Now calculate the values in the inequality from the definition:

$$E_{\vec{p1}}[S(\vec{p_1}, \vec{p_2}, \vec{h^3})] = \sum_{i=1}^{3} p_{1,i} \cdot S(\vec{p_1}, \vec{p_2}, \vec{h_i^3}) = 1.247 + 0 - 0.995 = 0.252$$

and

$$E_{\vec{p1}}[S(\vec{r}, \vec{p_2}, \vec{h^3})] = \sum_{i=1}^{3} p_{1,i} \cdot S(\vec{r}, \vec{p_2}, \vec{h_i^3}) = 1.247 + 0 - 0.687 = 0.56.$$

It is clear that the expected score for probability assessor 1 changed by using $\vec{r}$ instead of $\vec{p_1}$, hence the scoring rule is not strongly proper. It is, however, weakly proper.

Theorem 6.7 (Weak properness, The Bookmaker-Gambler scoring rule)**.** *The Bookmaker-Gambler scoring rule is weakly proper.*

Proof. First consider the definition of weak properness:

$$\begin{aligned} 0 &\geq E_{\vec{p1}}[BG(\vec{r}, \vec{p_1}, \vec{h^n})] = \sum_{i=1}^{n} p_{1,i} \cdot BG(\vec{r}, \vec{p_1}, \vec{h_i^n}) \\ &= \sum_{i=1}^{n} p_{1,i} \cdot \left(\frac{s_{r,i}}{p_{1,i}} - \frac{s_{p_1,i}}{r_i} \right) = \sum_{i=1}^{n} \left(s_{r,i} - s_{p_1,i} \frac{p_{1,i}}{r_i} \right) \\ &= 1 - \sum_{i=1}^{n} s_{p_1,i} \frac{p_{1,i}}{r_i} \end{aligned}$$

Let $j_1, \ldots, j_k$ be the indices where the stakes are placed which implies that each stake is $\frac{1}{k}$. This means that

$$\begin{aligned} 0 &\geq 1 - \sum_{i=1}^{n} s_{p_1,i} \frac{p_{1,i}}{r_i} = 1 - \sum_{i=1}^{k} \frac{1}{k} \frac{p_{1,j_i}}{r_{j_i}} \Leftrightarrow \\ 0 = 1 - \frac{k}{k} = 1 - \frac{1}{k} \sum_{i=1}^{k} 1 &\geq 1 - \frac{1}{k} \sum_{i=1}^{k} \frac{p_{1,j_1}}{r_{j_1}} \Leftrightarrow \\ \sum_{i=1}^{k} 1 &\leq \sum_{i=1}^{k} \frac{p_{1,j_1}}{r_{j_1}} \end{aligned}$$

Recall that the stakes are placed where $\frac{p_{1,i}}{r_i}$ is largest for $i = j_1, \ldots, j_k$, and because $\sum \vec{p_1} = \sum \vec{r} = 1$ then $1 \leq \frac{p_{1,i}}{r_i}$ for those is. Then $\sum_{i=1}^{k} 1 \leq \sum_{i=1}^{k} \frac{p_{1,j_1}}{r_{j_1}}$ is true which concludes Theorem 6.7.

Even though the Bookmaker-Gambler scoring rule is only weakly proper, it still resembles the problem domain. It should, however, be used with caution in other probability assessor evaluation domains but it fits the process of odds assessment, hence it is still used for tests later in this report.

6.2 Learning Probability Assessors

This section presents how probability assessors are learned in general terms. First, the true probability assessment is described. The true probability assessment is important because it is the probability assessment, all probability assessors should aim to output. How and why the dataset should be split up in order to get the desired results is described afterwards. This section concludes with a description of overfitting and why it should be avoided.

6.2.1 True Probability Assessment

Due to the large amount of unpredictable factors in a football match, the odds for a given wager are considered very hard to assess exactly. The odds are a representation of the underlying probability distribution, as described in Section 2.1.1 on page 5.

A true, or correct, probability distribution is considered to exist. Assume that an infinite number of matches, represented by instances, exist in the dataset, which also implies that an infinite number of identical matches existed. The correct probability assessment is defined as the frequency of the outcomes over groups of identical matches. If the correct probability is transformed into odds using a pay-back-percentage of 1, these can be considered the odds on which a given gambler does not care whether he has to buy or sell the wagers.

The true probability assessment can be represented as a vector $\vec{y}$ consisting of the probabilities for each possible option in the wager for a given match. $\sum_i y_i = 1$ and every element in the vector is one outcome of the wager.

The true function f is a function which maps the set of features to $\vec{y}$, such that f presents the correct probability assessment for the instance. f is not known, hence the task is to find a probability assessor C, i.e. a hypothesis of f in the space of all possible hypothesises, such that $C \in \mathcal{H}$. Let S be a set of training examples $\{(\vec{x}_1, v_1), \ldots, (\vec{x}_m, v_m)\}$, where $\vec{x_i}$ is a vector of values on the form $\langle x_{i,1}, \ldots, x_{i,n} \rangle$, and $x_{i,j}$ is the jth either continous or discrete feature in the ith training example. v_i is the class variable of

the ith training example, indicating how the wager ended on the already played match. $C(\vec{x}) = \vec{z}$ will be probability assessor C's estimation of the probability distribution $\vec{y}$ for the features $\vec{x}$. Note that the probability assessor C may have a higher probability on the correct outcome than f on a given match. However, if enough matches with the same $\vec{x}$ values were present, the outcome of the wagers would be distributed like f does.

6.2.2 Utilise the Training Data

When the error of a given probability assessor needs to be determined, the dataset is usually partitioned into two sets: the learning set, L, and the validation set, V. These sets are described in Section 5.4 on page 51. But the learning set often has to be split again because it is used for parameter tuning. The evaluation of the tuned settings should be performed by learning a model using the settings on L_L and testing the model on L_T where $L_L \cap L_T = \emptyset$.

Holdout validation is the simplest kind of validation. A given number of instances are randomly chosen from L, these instances form the test set, L_T, and the remaining instances form the actual learning set, L_L. Usually, approximately a third of the instances are used in L_T [DHS01].

A problem with the splitting is that each instance is used only for either learning or testing, so the available instances are only used for one action each. This implies that the precision of the error decreases compared to the case where all instances are used for both learning and testing. To avoid this reduction in precision, a process called cross-validation is used. There are two common approaches to performing cross-validation, [TSK97]:

- k-fold cross-validation where the dataset is partitioned at random into k subsets. When the score is calculated, one subset is chosen as L_T and the $k-1$ remaining subsets are used as L_L. This process is repeated k times, until all the subsets have been used exactly once as L_T. The error can be found as the average score over the k subsets.

- Leave-one-out cross-validation is k-fold cross-validation with k chosen as the number of instances in the dataset. This implies that every instance forms its own subset.

From the above variations of validation, leave-one-out cross-validation is the one that allows the greatest number of instances to be used when making a probability assessment. It is the one which achieves the highest accuracy in estimating the score. The downside of leave-one-out cross-validation is that it is also the one which generates the most calculations.

6.2.3 Overfitting

This section is based on [Geu02], and [Mit97]. Overfitting occurs when the learned model tries to fit the learning data too much, such that it becomes extremely adept at making classifications for the instances in the learning set but bad at making classifications for other instances. Note that the notion of overfitting is described using a problem setting with a classification problem as this is more intuitive to understand, but it still applies to a probability assessment problem setting.

Definition 6.8 (Overfitting (from [Mit97]))**.** *Given a hypothesis space $\mathcal{H}$, a hypothesis $h \in \mathcal{H}$ is said to overfit the training data if there exists some hypothesis $h' \in \mathcal{H}$, such that h has a smaller error than h' on the training data but h' has a smaller error than h over the entire distribution of instances.*

Another related term is underfitting which occurs when the model does not fit the data enough. These terms can also be described in the context of bias and variance. Intuitively, these terms can be described as when a classifier partitions the instance space into regions where each region is assigned a class. When a new instance needs to be classified, the model determines which region the instance fits into and then assigns the class of the region to the instance. When these regions are too large or general, the accuracy of the fit generally decreases, increasing the error rate. This effect is called bias, and is shown on the left in Figure 6.1. When the regions are too small or the complexity of the regions is too great, the probability that the regions are labelled with an incorrect class is increased, again increasing the error rate. This effect is called variance, and is shown on the right in Figure 6.1. The figure is created using k-Nearest Neighbor, hence k refers to the number of nearest neighbours the algorithm takes into consideration before it makes its classification.

There exists a trade-off between bias and variance, or between an instance space with too simple regions and an instance space with too complex regions. This is shown in Figure 6.2. The optimal fitting is the point where the lowest error is measured, and also where neither over- nor underfitting occurs. Overfitting is present to the right of the optimal fitting as the complexity is increased, whereas underfitting is present to the left of the optimal fitting as the model turns simpler.

6.3 Test Plan

This section presents the selection of algorithms and the test phases.

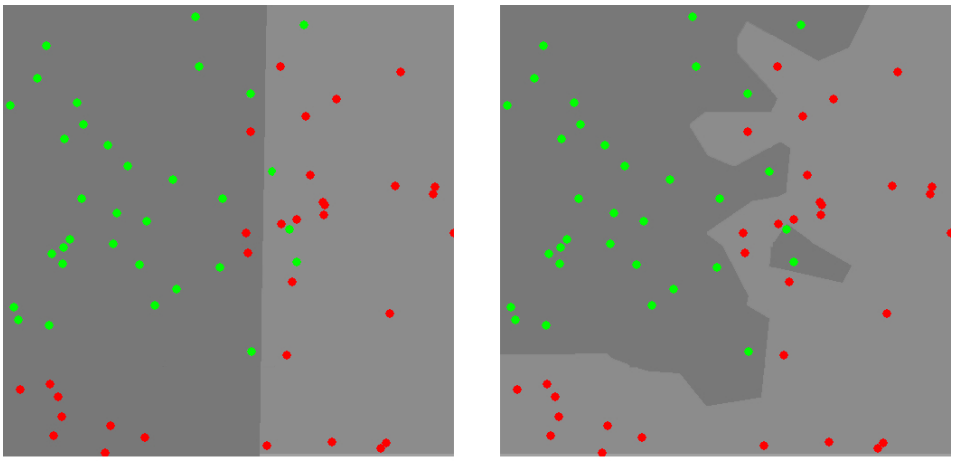

Figure 6.1: The intuition behind bias and variance, with bias shown left and variance shown right.

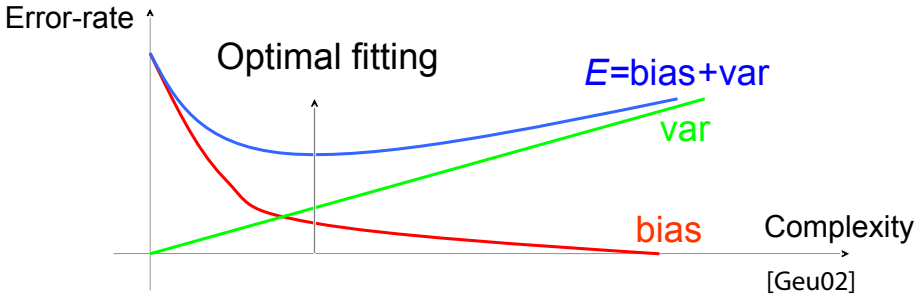

Figure 6.2: The trade-off between bias and variance, with the error-rate shown as a function of the complexity.

6.3.1 Selection of Algorithms

The *no free lunch theorem*, described in [WM97], states that no single learning algorithm fits all problem domains. For that reason, different learning algorithms should be applied to the data.

An eager learner creates the probability assessor as soon as the training data is available, implying that when it encounters a test instance, it is ready to make a probability assessment based on that instance. Lazy learners do not evaluate the training instances before a probability assessment of a test instance is needed, so it needs to make the probability assessor before each probability assessment, implying that the probability assessment process takes a longer time. An advantage of the lazy learners is that after an instance is probability assessed and the outcome is observed, the instance can easily be a part of the learning data.

The major difference between the two classes of learners is that since a lazy learner produces a new probability assessor for each new instance which needs a probability assessment, the approximation is customised to the exact instance. This implies that the approximation is local. An eager learner must choose the approximation in advance before the instance is known, which implies that the approximation is global [Mit97].

The first learning algorithm is a modified version of the decision tree algorithm which instead of the normal classification produces a probability assessment. The decision tree algorithm is an eager learning algorithm which partitions the instance space into regions where the same probability assessment is made in each region. The details are presented in Chapter 7.

The k-Nearest Neighbor algorithm is a lazy learner and is presented in Chapter 8. The algorithm uses the nearest neighbours measured by Euclidean distance to make its probability assessment which, is done at evaluation time making it a lazy learner. An alteration of k-Nearest Neighbor, where the best k value is found automatically, is also presented.

As described in Section 2.2 on page 11, the Dixon-Coles approach fits the problem domain well, hence this approach is presented in Chapter 9. In contrast to decision trees and k-Nearest Neighbor, the Dixon-Coles approach only uses one feature, namely goals.

Ensemble methods are used to combine a number of base probability assessors, like decision trees, into a stronger combination than the base probability assessors themselves. Both ensembles of base probability assessors of one type and ensembles of different types of base probability assessors are presented and evaluated. Ensemble methods are presented in Chapter 10.

6.3.2 Test Phases

The test consist of three phases. First, the learning algorithms and the necessary alterations to fit the problem domain are explained in each of the four chapters.

The parameter tuning and model learning is the second phase. The models are first learned using the learning set, L_L, consisting of the matches played before the 1st of July 2006 which is 396 matches. This also includes the individual parameters like the k value in k-Nearest Neighbor but also feature selection which is done differently in the learning algorithms. The tuning is based on the logarithmic score, see Section 6.1.1, obtained for the different settings. This phase ends up in a test on the first test set, L_T, consisting of 107 matches played between the 1st of July 2006 and the 1st of January 2007. The second phase is described individually in each chapter. After the tuning and testing, one to three different settings are chosen for each learning algorithm.

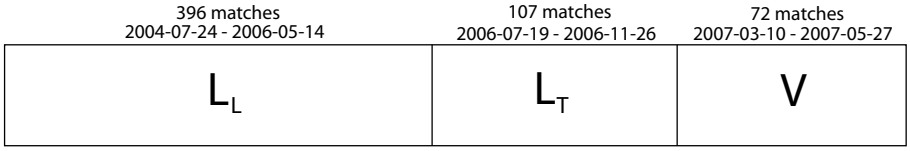

Figure 6.3: The entire dataset is divided into three subsets.

The final evaluation is the third phase. The evaluation is performed in Chapter 11 and uses matches neither in L_L nor in L_T to evaluate each of the different learning algorithms and settings. This dataset will be named the validation set, V, and consist of 72 matches played after the 1st of January 2007. The partition of the dataset into the three subsets is shown in Figure 6.3. The evaluation is both as a bookmaker and as a gambler. The details for the final evaluation are presented in Section 11.1 on page 119.

Chapter 7

Decision Trees

A decision tree is a graph or model, in which each internal node tests a feature and each leaf node assigns a classification or probability assessment. The decision tree can, given a test instance, make the probability assessment, only by observing the most important features. It determines which features are most important when building the tree. This section is based on [HK01] and [Mit97].

The decision tree classifier is an eager learner, implying that it builds the classifier before it knows anything about the instances that need to be classified. In the basic case, the decision tree algorithm is a greedy agorithm which constructs the decision tree using a top-down recursive divide-and-conquer approach. Most decision tree algorithms are designed for cases where the class feature is discrete, e.g. ID3, C4.5. Decision tree algorithms for regression also exist but they are not described because they do not fit the project needs.

The probability assessment for an instance is looked up in the tree and for each node the split of the feature is compared to the feature in the instance. This continues down to the next node until a leaf node is reached which then returns the probability assessment as the probability assessment for the instance.

Two of the main benefits by using a decision tree are:

- It is not necessary to perform feature selection before learning, as the feature selection is part of the decision tree building.
- Once a decision tree is made, it is very easy to interpret compared to e.g. k-Nearest Neighbor. Once a decision tree is built, a human can easily interpret it but most importantly, classification or probability assessment is very quick and easy.

This chapter first introduces how feature selection can be performed followed by how feature selection is used to build the decision tree. In this project, a modified version of ID3 is used, this version and the implementation of it is described, and finally the test results are presented using this implementation.

7.1 Feature Selection

Information gain is the method used for feature selection in ID3. It is a measure used to select which feature the next split should be made on, hence it is also called an *attribute selection measure* or a *measure of the goodness of split*. The feature with the highest information gain is chosen, this feature is also the feature with the greatest entropy reduction.

Entropy describes the purity of a given dataset. The lower the entropy of a dataset, the higher the purity. The formal definition is shown in Definition 7.1.

Definition 7.1 (Entropy)**.** *Let a dataset, $S = (x_1, x_2, \ldots, x_n)$, be a distribution of the possible outcomes of the class variable and $x = \sum_{i=1}^{n} x_i$, and n the number of possible values of the class feature, then*

$$Entropy(S) = -\sum_{i=1}^{n} \frac{x_i}{x} \cdot \log_2 \frac{x_i}{x} \tag{7.1}$$

Is should be mentioned that when the entropy is calculated, $0 \cdot \log_2(0) \equiv 0$. Figure 7.1 shows how the entropy of a dataset evolves, when the dataset has a fixed number of instances and the class feature is binary, with classes A and B. The X-axis shows the proportion of instances that is classified as A. Note that the proportion of instances classified as B is dependent on how many is classified as A.

The expected entropy after the split at some feature is calculated using Definition 7.2.

Definition 7.2 (Expected Entropy)**.** *Let S, x, and n be defined as in Definition 7.1. Let $S_v = (x_{1,v}, x_{2,v}, \ldots, x_{n,v})$, $x_{i,v}$ be the number of instances with the ith class variable and feature value v, and $x_v = \sum_{i=1}^{n} x_{i,v}$. Furthermore, let $Z = \{v_1, \ldots, v_m\}$ be a feature with possible values $v_1, \ldots, v_m$. After the dataset is split at feature Z, the expected entropy can be calculated as*

$$ExpectedEntropy(S, Z) = \sum_{v \in Z} \frac{x_v}{x} \cdot Entropy(S_v) \tag{7.2}$$

Information gain calculates expected reduction in entropy by splitting the dataset at a given feature, so the difference in entropy before and after a split is measured. Information gain is defined in Definition 7.3.

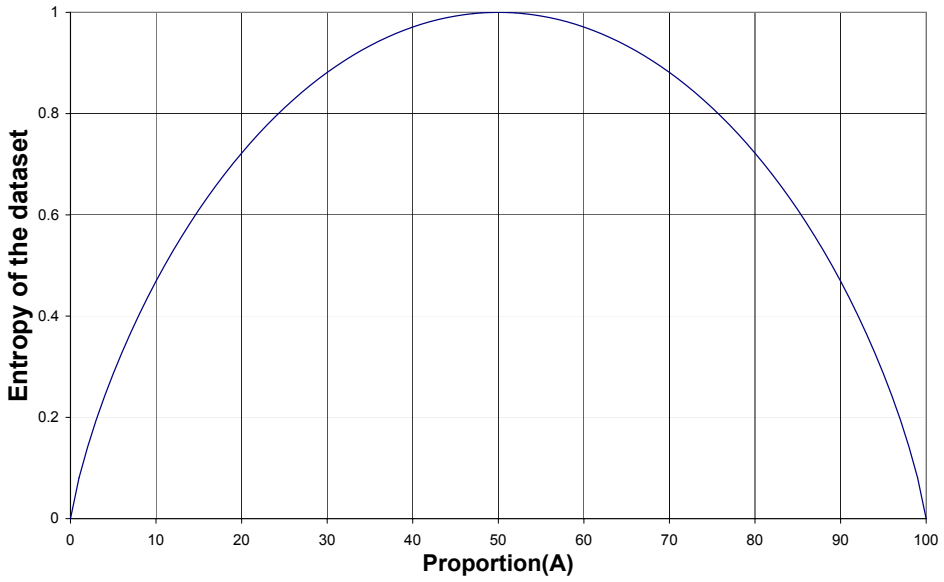

Figure 7.1: The entropy function relative to the binary classification, (A,B), where Proportion(A) is the proportion of instances labelled A.

Definition 7.3 (Information Gain)**.** *Let Entropy(S) be the entropy of a dataset, S, prior to splitting it, and ExpectedEntropy(S,Z) be the expected entropy of S, after splitting it on feature Z, then*

$$Gain(S, Z) = Entropy(S) - ExpectedEntropy(S, Z) \tag{7.3}$$

One potential problem with using information gain is that, it does not take the number of possible values into account, implying that features with data like dates or ids have a very high information gain as they most likely will make a perfect split on the dataset.

7.2 Building a Decision Tree

One of the basic decision tree generation algorithms is the Iterative Dichotomiser 3 (ID3) algorithm. The building of a decision tree will be explained in the context of how ID3 would build a decision tree.

The ID3 uses an approach where it starts with the empty tree and incrementally grows the tree more complex. The ID3 algorithm uses information gain to determine which feature the next split should be made on. The ID3 algorithm does not backtrack, once it determines which feature to use at a given node the decision is final. This leads to the danger of finding a local maximum and not the global maximum.

The ID3 algorithm has an inductive bias, that is shorter trees are preferred over taller trees and features with a high information gain are preferred close to the root. This is caused by the ID3 algorithm growing the tree from the empty tree and stopping when it reaches the first tree that is consistent with the training data. A reason why a complex tree is not necessarily optimal is that one or several erroneous instances, noise in the data, would render the classification or probability assessment erroneous. This can lead to overfitting, as described in Section 6.2.3 on page 66.

If the decision tree were fully it would grow until each leaf node had an entropy of 0 leading to a high complexity. To prevent this, either the growing of the tree can be stopped before the tree fits the data perfectly, this is called prepruning, or pruning can be applied after the decision tree is built, this is called postpruning.

If the decision tree is to be postpruned the learning set, L, is spilt into two datasets, L_L and L_T [Mit97]. The decision tree is then grown from L_L and can be pruned using L_T. There are several ways of postpruning the decision tree, reduced error pruning is one way of doing this. In this approach, the complete tree is built, using L_L. When the tree is built all nodes in the tree are considered candidates for pruning. This implies that one at a time each node is removed, implying that any subtree of that node is also removed. The outcome frequency of any subtree is assigned to the node in question. The accuracy on L_T is calculated, when this exact node is missing from the tree. If the accuracy is no worse than the accuracy of the original tree, this node can be removed. The major drawback of this approach is that it involves splitting the dataset which, given a small dataset, can make the decision tree worse than if the entire dataset was used.

There are a number of ways of prepruning the tree: setting a lower limit on the information gain required to make a split, setting a lower limit on the size of the nodes that can be split, or setting a lower limit on the size of nodes. However, choosing the optimal threshold is difficult, a high threshold yields a very simple decision tree with high bias whereas a low threshold yields a very complex decision tree with high variance.

7.3 Implementation

The implementation chosen for this project resembles ID3 but differs on some points. ID3 is used in this project even though the features are continuous, hence the features are discretised. This is done by making tertiary splits, and then calculating the expected information gain for each possible split. A possible split is one that splits the dataset into three non-empty datasets, where the size of each set is above a given lower limit.

Tertiary splits are assumed as there are three possible outcomes (1,X,2), hence intuitively if a perfect split existed the dataset could be divided into three datasets with entropy 0, by using a single feature. Another benefit of using tertiary splits is that since all features have three possible values, the potential problem associated with information gain, concerning features with many different values, is removed.

When making probability assessments growing a large tree would lead to a very low number of instances in the leaf nodes, resulting in little flexibility in the probability assessments and increasing the variance. If a leaf node only contained 4 instances, the lowest possible probability on a outcome, except 0, is 0.25. Overfitting in the implementation prevented by stopping the growth of the tree before it terminates completely.

Normalisation is not necessary as the difference in the values of the features does not matter, due to the built-in feature selection algorithms.

The implemented ID3 algorithm is shown in Algorithm 71 on page 77. The ID3 algorithm takes *dataset*, which is the learning set, *features* which are the unused features, and *minSize* which is the minimum size any split can have. The ID3 algorithm starts off by, creating a new node, in line 2. In line 3, it is checked whether all instances in the dataset have the same label. If so, the split is not performed and the algorithm continues to line 4 where the root node is assigned the frequency of the labels of the instances in the dataset. In line 6 and 7, S_b is initialised and *bestEntropy* is set to infinite. S_b is used to keep track of the best datasets after splits and *bestEntropy* is used to keep track of which split achieves the lowest expected entropy after the split.

Lines 8-25 consist of two nested for loops which for all pairs of feature values discretisise a given feature. This results in three new datasets, S_1, S_2, and S_3, with instances dictated by the new interval which is done in lines 10-12. In line 13, the set S that consists of S_1, S_2, and S_3 is created.

Line 14 checks whether all of the datasets contained in S are above *minSize*. If they are not, the loop is finished and another set of values are extracted in line 9. If they are above *minSize*, the expected entropy after this split is calculated in line 15. If the expected entropy after this split is performed is lower than the current best expected entropy, *bestEntropy*, the expected entropy, the feature, the two values, and the datasets are recorded in lines 17-21.

Line 26 tests whether a new split was found where all subsets are larger than *minsize*. If not, the root node is assigned the probability distribution of the dataset. If on the other hand a better split was obtained, the set *features* is updated in line 29 by removing the feature that had the lowest expected entropy. The for loop in lines 30-32 adds a new child to the decision tree for each dataset. This is done via the `addChild` function which takes a decision

tree and adds this as a child to the current decision tree root. The `addChild` function calls the `ID3` function, implying that the `ID3` function is recursive.

7.4 Test Results

The learning set, L_L, consisted of 396 matches from the 04/05 and 05/06 season, and the test set, L_T, consisted of 108 matches from the 06/07 season. Furthermore, all features were used, implying that also the overall features, described in Section 5.2.2 on page 49, were used.

The implemented version of ID3, described in last section, can be changed in two ways; the *minsize* value and the *window* size.

One combined test was made to determine which values are best for *minsize* and *window*. All values for *minsize* between 0 and 80 were tested in steps of size 2. The upper limit of 80 is chosen because at every node the dataset is split in three, this means that at least 160 matches are used in the root for two of the splits, while the third split will have a maximum of 236 matches, leaving no room for any further splits. This causes any trees with *minsize* above 80 to only use one feature. For each *minsize*, all *window* sizes between 1 to 12 were tried. A plot for all the average logarithmic scores for the decision trees is shown in Figure 7.2.

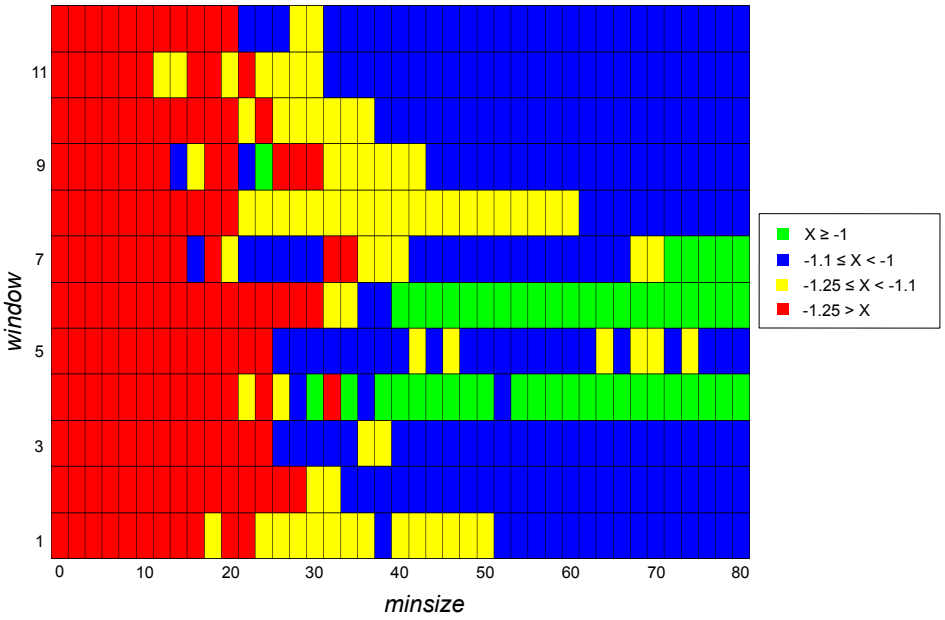

Figure 7.2: A plot of the the average logarithmic score as a function of the *minsize* value and the *window* size.

Algorithm 7.1 The implemented ID3 algorithm for building a decision tree.

```
1: function ID3(dataset, features, minSize)
2:     rootNode ← new NODE()
3:     if dataset.ISALLLABELSSAMEKIND() then
4:         rootNode.probDist ← dataset.GETPROBDIST()
5:     else
6:         S_b ← ∅
7:         bestEntropy ← ∞
8:         for all feature ∈ features do
9:             for all {v_1, v_2 ∈ feature.allValues| v_1 < v_2} do
10:                S_1 ← {i ∈ dataset | i.GETVALUE(feature)≤ v_1}
11:                S_2 ← {i ∈ dataset | v_1 < i.GETVALUE(feature)≤ v_2}
12:                S_3 ← {i ∈ dataset | v2 < i.GETVALUE(feature)}
13:                S ← {S_1, S_2, S_3}
14:                if ∀s ∈ S.|s| ≥ minSize then
15:                    expectedEntropy ← Σ_{j=1}^{|S|} (|S_j|/|dataset|)·ENTROPY(S_j)
16:                    if expectedEntropy < bestEntropy then
17:                        bestEntropy ← expectedEntropy
18:                        rootNode.feature ← feature
19:                        rootNode.value_1 ← v_1
20:                        rootNode.value_2 ← v_2
21:                        S_b ← {S_1, S_2, S_3}
22:                    end if
23:                end if
24:            end for
25:        end for
26:        if S_b = ∅ then
27:            rootNode.probDist ← dataset.GETPROBDIST()
28:        else
29:            features ← features\{rootNode.feature}
30:            for all s ∈ S_b do
31:                rootNode.ADDCHILD(ID3(s, features, minSize))
32:            end for
33:        end if
34:    end if
35:    return rootNode
36: end function
```

The figure shows that *window* sizes 4 and 6 are better than the other possible *window* sizes, and that a *minsize* value of at least 40 is preferable. It is clear that all decision trees with a *minsize* value of 20 or below are likely to be worse than an average logarithmic score of -1.25, implying that the decision tree is unusable in practice.

The best average log scores from the decision trees are found and shown in Table 7.1. The best decision tree has a log score of -0.9655, a *minsize* of 28 or 30, and a *window* size of 4. That tree is shown in Figure 7.3 on page 79. This decision tree uses the features *Overall Made a keypass*, *Overall Finished with left foot in open play shot were blocked*, and *Overall Crossed to team mate on corner*. The category "Made a keypass" was also used by the k-Nearest Neighbor algorithm and it is used in all the best decision trees. Other features that are used deal with either corners or offensive passes.

Note that there are two identical decision trees performing best which is caused by no leaf nodes in the decision tree with *minsize* of 28 having less than 30 instances. This can be observed in Figure 7.3 on page 79, where Node 7 has the least instances, 31.

Note further that due to the way the features are constructed, as described in Section 5.2.2 on page 49, the first splits are -17 and 8.5, and the first dataset is primarily away wins. This implies that the higher the value of the feature *Overall Made a keypass*, the more likely it is that the match will be a home win. This was also what could be expected, as a positive value in this feature indicates that the home team has made more keypasses than the away team over the last few matches. This pattern is not consistent with the distributions of the datasets created by splitting *Overall Finished with left foot in open play shot were blocked* but it is when splitting *Overall Crossed to team mate on corner*.

From the performed tests, it can be observed that with *window* sizes of 4 or 6 and a *minsize* above 40, a log score around -1 could be expected. Such a logarithmic score is comparable to the bookmakers logarithmic scores, implying that a useful decision tree algorithm might be built. Though there seems to be some instability in the probability assessors, due to the use of decision trees, implying that ensemble methods could be useful. Ensemble methods are further described in Chapter 10.

log score	*minsize*	*window*	features
−0.965505	28, 30	4	Overall Made a keypass, Overall Finished with left foot in open play shot were blocked, Overall Crossed to team mate on corner
−0.968601	80	7	Overall Made a keypass
−0.972528	72, 74	7	Overall Made a keypass
−0.977723	76, 78	7	Overall Made a keypass
−0.978193	78, 80	4	Overall Keypassed from set piece
−0.982402	40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80	6	Overall Made a keypass
−0.98514	24	9	Overall Finished off target, AwayFor Made a keypass, Overall Took a corner
−0.988574	34	4	Overall Finished with left foot in open play shot were blocked, Overall Made a keypass, HomeFor Completed offensive short pass
−0.989046	50	4	Overall Made a keypass, Overall Uncompleted offensive long pass
−0.992529	38, 40, 42, 44, 46	4	Overall Made a keypass, Overall Uncompleted offensive long pass
−0.992928	48	4	Overall Made a keypass, Overall Uncompleted offensive long pass
−0.99785	54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76	4	Overall Made a keypass

Table 7.1: The decision trees with average logarithmic score higher than -1.

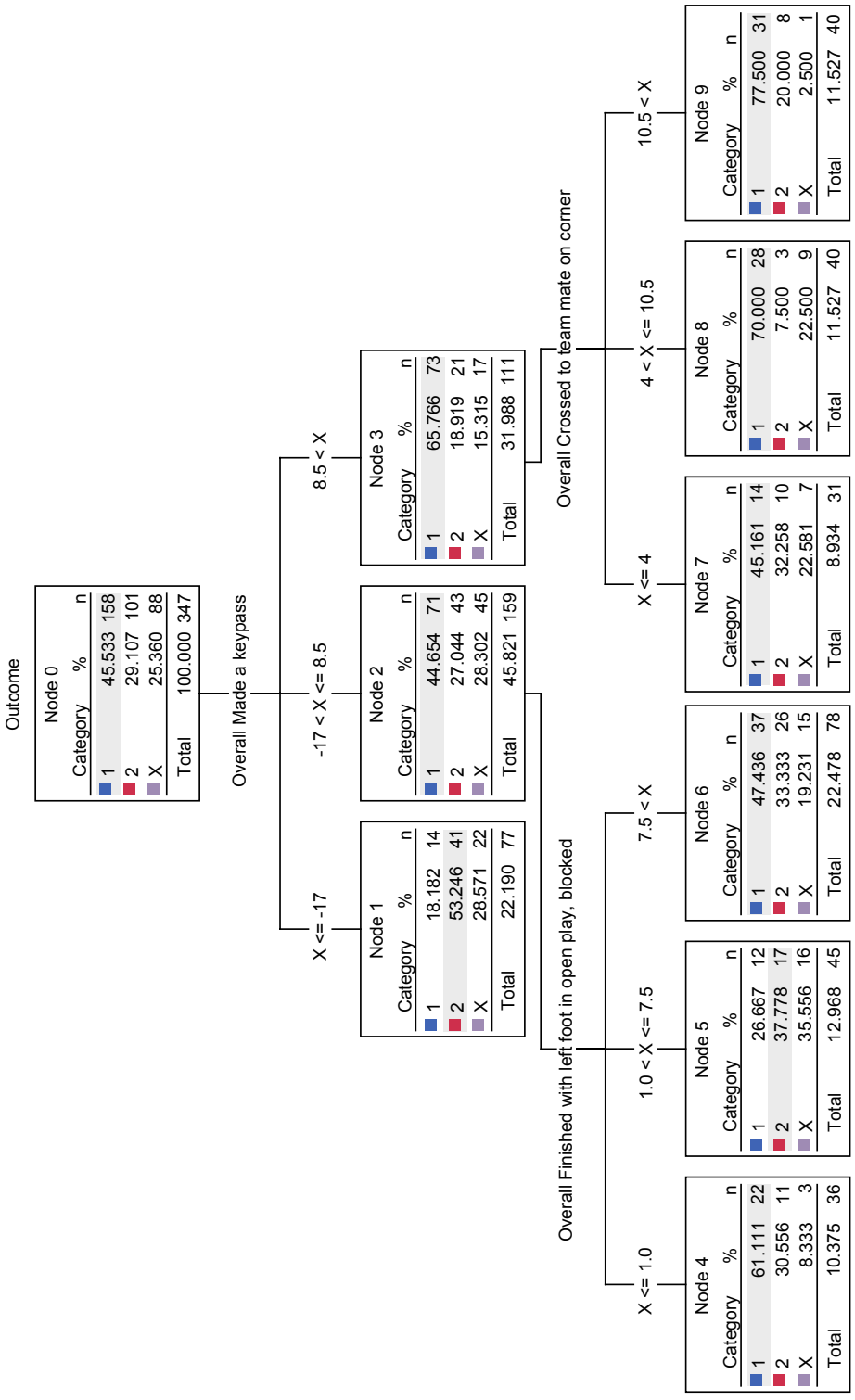

Figure 7.3: The best decision tree, with regard to logarithmic score, found in the test.

Chapter 8

k-Nearest Neighbor

This chapter first presents an overview over the k-Nearest Neighbor and how it makes probability assessments. A feature search algorithm can be used to reduce the number of features which is necessary when a lot of features are available, so such an algorithm is presented. The optimal k value needs to be determined which in this project is done by testing all different k-values on the learning data. Lastly, the initial test results which determined the settings used in the final test on fresh data are presented. The chapter is based on [Mit97], [HMS01], and [Lar05].

8.1 Making Probability Assessments

The k-Nearest Neighbor algorithm is from the class of lazy learners, which in contrast to the class of eager learners, e.g. decision trees, do not evaluate the training instances before a probability assessment of a test instance is needed.

k-Nearest Neighbor builds, as all classifiers, on the assumption that future instances will be similar to previous instances. Similarity is calculated using the Euclidean distance measure as described in Section 4.4 on page 36. When the k-Nearest Neighbor algorithm has to make a classification or a probability assessment of a new instance, it starts by calculating the distance from the new instance to all instances in the learning set, L, and then the k nearest instances are found. If a classification is needed, the majority of outcomes among the neighbours is used as the final classification. If a probability assessment is needed, it is done simply by using the frequency of the different outcomes as the probability. E.g. in an instance with three potential outcomes and $k = 20$, the probability assessment could be $\langle \frac{12}{20}, \frac{6}{20}, \frac{2}{20} \rangle$. Since the k-Nearest Neighbor algorithm requires no initial

build time, obtaining a probability assessment from the k-Nearest Neighbor algorithm takes longer than using the decision tree algorithm. This is caused by the fact that the distances between the instance in question and all instances in L are calculated.

8.2 Feature Search Algorithm

k-Nearest Neighbor uses all features in the dataset, in contrast to e.g. decision trees which only select a few features from the dataset. Hence, the amount of data needed increases exponentially with the dimensionality if the accuracy level should be maintained. This problem is the curse of dimensionality, described in Section 5.2 on page 48.

The curse of dimensionality can be dealt with in two ways: either by reducing features by aggregating existing features and removing the original features, or simply by removing some of the original features from the dataset. Both approaches are explained in Section 5.2. Another potential problem is that irrelevant features dominate the distance between neighbours, hence those features should be removed from the dataset. Also highly correlated features dominate other features, so it should also be considered removing some of the highly correlated features.

As described in Section 5.2.3, the optimal feature subset cannot be found in polynomial time using a wrapper approach since the power set of features must be tested. This implies that if the number of features in the feature set is large, the algorithm will use very long time.

There are a number of different schemes which could be used to traverse the complete lattice which is formed by all possible feature subsets: depth-first search, breadth-first search, or best-first search. Even though there are a very large number of features available, it is expected that a simple model would perform best. This implies that the size of the subset of features which needs to be selected is expected to be low. Hence, a depth-first search will search through too many feature sets which are not expected to be useful, as it will traverse all the way to the bottom of the complete lattice before returning to the top. If breadth-first search is used, the search algorithm would have to search through a huge number of small useless subsets only consisting of features with low information. A best-first search greedily searches through the lattice in search of the best subset. This implies that quality models can be found more quickly with a best-first search [HMS01]. Hence, the feature search is performed with a best-first search.

Another intuitive reason for using a best-first search is that it is expected that if a feature subset performs well, then that subset combined with another feature may also perform well.

Some evaluation of each feature subset must be done in order to determine which subset is the better. This is done by building a probability assessor using the feature subset and applying a scoring rule which can evaluate its usefulness. The algorithm can stop after a number of visited subsets have provided no gain or continue to run until all combinations are tested. In either case, the best subset can always be extracted even though the algorithm continues to run. The algorithm is shown in Algorithm 8.1.

Algorithm 8.1 The feature search algorithm.

```
1: function featureSearchAlgorithm(features, scoreF, maxWOgain)
2:     rankings ←new SortedSet(sort by: unvisited, best score)
3:     rankings.add(−∞, ∅, unvisited)

4:     woGain ← 0
5:     bestScore ← −∞
6:     while unvisited subsets in rankings and woGain < maxWOgain
   do
7:         bestCombi ← rankings.getBestCombination()
8:         woGain ← woGain + 1
9:         for all f ∈ features\bestCombi do
10:            newCombi ← {f} ∪ bestCombi
11:            if newCombi ∉ rankings then
12:                score ← scoreF.getScore(newCombi)
13:                if score > bestScore then
14:                    bestScore ← score
15:                    woGain ← 0
16:                end if
17:                rankings.add(score, newCombi, unvisited)
18:            end if
19:        end for
20:        rankings.setVisited(bestCombi)
21:    end while

22:    return best scoring combination in rankings
23: end function
```

The algorithm takes a set of features, *features*, a score function, *scoreF*, and an integer indicating when the algorithm should stop searching, *maxWOgain*, as parameters.

In line 2, a sorted set is created which first sorts the unvisited items at top and then order those items by the best score. In line 3, the most simple feature combination is added, the one without any features. This combination is marked as **unvisited** because the algorithm has not visited that combination yet. The score for the combination is set to the worst possible

such that all other combinations are regarded as better.

From lines 4–21 the subsets of features are searched. First, a counter, *woGain* which counts how many combinations have been visited without gain, is initialised to 0. The best score found so far is set to $-\infty$ such that any other score is regarded as better.

The loop in line 6 for searching subsets will run as long as there is at least one not yet visited subset. The loop will also end if there has been a number of visited combinations without gain, more exactly the number specified as a parameter to the algorithm. Note that the number of visits is counted in terms of subsets which have been the best available at the time they were visited.

In line 7, the best currently not visited subset is extracted, and in line 8 the counter for nodes visited without gain is incremented by one. Note that if a better subset is found when visiting a subset, the counter is reset.

In line 9, all features not in the current subset are extracted, such that they in line 10 can be added to the current subset one by one to yield new combinations. In line 11, it is checked that the combination has not already been explored. In line 12, the evaluation rule, supplied as a parameter to the algorithm, is used in order to evaluate the current combination. For instance, this evaluation rule can build a k-Nearest Neighbor probability assessor and then evaluate it to get a score for the combination. In line 13, it is checked whether the new score is better than anyone seen before. If so, the *bestScore* variable is set to the new best score and the number of visited subsets without gain is reset. In line 17, the newly explored combination is added to the *rankings* set with its score, and it is marked as **unvisited** so that the algorithm can test it in combination with other features later on.

In line 20, the combination currently being visited is set to **visited**, such that it is not visited again.

The algorithm ends by returning the best combination in line 22.

Note that an external application, provided that some concurrency issues are resolved, can always find the best known feature combination in *rankings*. This implies that even though the algorithm can run for a long time, it is still able to deliver results before it terminates.

A distributed version of the algorithm can easily be made. The loop starting in line 9 can make a thread for each combination, provided that the `getScore` function can be executed in parallel. When all threads are finished running and the score is obtained for each combination, the algorithm can continue in line 13 by finding the best new combination and making sure that each of the new found combinations are added to *rankings* like in line 17. The distribution of the algorithm can be very useful because it usually has a long running time if there are a large number of features in the feature set.

8.3 Optimal k Search

When k is increased, the variance is reduced and the bias of the probability assessments is increased as the output gets closer to the overall probability distribution of the class variable in the entire training set. If k is too small, the variance would be too great, hence the classifier would be unstable. If k is too high, some of the instances identified as being nearest neighbours could potentially be far from the actual instance in question. This indicates that there exists a trade-off between bias and variance. The best method for finding the optimal k is a data-adaptive one, implying that k needs to be adapted to each particular dataset via a trial-and-error approach [HMS01], [Lar05].

Optimal-k k-Nearest Neighbor is an algorithm for determining the best k value for a dataset. It is applied to the learning data and returns the best k value for the the data based on what score the different instances get with different k values. By applying Optimal-k k-Nearest Neighbor to learn the k value, the process becomes eager because the k value has to be found before probability assessments are made for the test instances. The normal k-Nearest Neighbor algorithm is still needed when probability assessments are made for new instances. Note that the Optimal-k k-Nearest Neighbor algorithm is only used to determine the optimal k value, implying that once the optimal k is determined this k value is used as a setting for the regular k-Nearest Neighbor algorithm.

Algorithm 8.2 k-Map k-Nearest Neighbor.

```
1:  function kMapKNN(trainingData, testInstance, scoringRule)
2:      dist ← new SortedMap()
3:      for all i ∈ trainingData do
4:          dist.put(EuclideanDistance(i, testInstance), i)
5:      end for

6:      pam ← new ProbabilityAssessment()
7:      kMap ← new Map()
8:      for k = 1 to |trainingData| do
9:          instance ← dist.firstValue()
10:         dist.remove(instance)
11:         pam.add(instance.cv())
12:         score ← scoringRule.getScore(testInstance.cv(), pam)
13:         kMap.put(k, score)
14:     end for

15:     return kMap
16: end function
```

Optimal-k k-Nearest Neighbor uses the kMapKNN function, shown in Algorithm 8.2, as part of the algorithm. In line 2 of the kMapKNN function, the distance map, *dist*, is initialised. The key values of *dist* are the distances between *testInstance* and the entries value, which is the instance currently in question. Since the sorted map sorts its content according to the key values, the key with the smallest value, that is the instance from the learning data which is closest to the test instance, is always obtainable. In lines 3–5, the learning instances are put into the map. Note that EuclideanDistance is a function which determines the Euclidean distance between the two instances as described in Section 4.4 on page 36.

In lines 6–14, the probability assessments are made and evaluated for each possible value of k. The ProbabilityAssessment constructor creates a class where a number of class variables of the same type, e.g. class variables for outcomes, can be added. Then the probability assessment can be obtained from the class. e.g., if a home win and an away win is added to a newly constructed ProbabilityAssessment, the ProbabilityAssessment class would return 0.5 chance for home win and 0.5 chance for away win.

Each possible k-value is iterated over in line 8. In line 9, the instance closest to the test instance is extracted and this instance is removed from the map in line 10, such that the next closest instance is extracted in the next iteration. In line 11, the class variable of the just extracted instance is added to the probability assessment which changes the probability assessment accordingly, as mentioned above.

In line 12, the score is obtained from the scoring rule which was given as parameter. The scoring rule must have a getScore function such that the score for a given class variable and probability assessment can be obtained. In line 13, the current k value and the score are added to the map containing all k values and scores. The algorithm ends in line 15 by returning the map with k values and scores.

Optimal-k k-Nearest Neighbor works by using leave-one-out cross-validation to create a k map for each instance using the kMapKNN function, such that a score for each instance and k value is available. Then for each k value, the average score over all instances is calculated, and the best average score and the best k value are returned from the Optimal-k k-Nearest Neighbor algorithm. Note that this approach uses memory quadratically in the size of the dataset, hence it will not be useable in all scenarios. However, the dataset in this project is not very large which implies that the approach can be used.

8.4 Test Results

This section presents how the settings for the test are determined. The test is made in several phases. The first four phases use matches from two

seasons and cross-validation to find the best settings for k-Nearest Neighbor and result in a selection of settings. These phases consist of a draft selection of features, selection of window size, selection of normalisation and a final selection of features. The four phases all use the Optimal-k k-Nearest Neighbor algorithm presented in Algorithm 8.2 to determine the optimal k value.

The final phase uses half a season of not previously used data to determine whether the found model is overfitted. A simple model is also tested in that phase for comparison and the best of the two models is used in the final evaluation against the other types of learning algorithms.

8.4.1 Determining the Feature Subsets

First, the best features are determined. This is done using the feature search algorithm presented in Algorithm 8.1 and the logarithmic scoring rule presented in Section 6.1.1 on page 57.

In Table 8.1 the results four tests are shown. The standard settings in all tests are window size four, as advised in [sta], and min-max normalisation.

Id	k	Features	Log. score
1	25	Assisted from a set piece, Finish with right foot from set piece shot were blocked, Finished with head but it was blocked, Free kick shot off target, Keypassed from set piece, Made an error, Received a red or 2nd yellow, Scored with right foot from set piece, Scored on direct free kick	−0.9983
2	25	Assisted from a set piece, Finished with other in open play but ball was blocked, Finish with right foot from set piece shot was blocked, Made an error, Free kick shot off target, Scored with right foot from set piece, Finished with head but it was blocked, Keypassed from set piece, Scored on direct free kick, Received a red or 2nd yellow	−0.9995
3	23	Made a keypass, Made an error, Scored	−1.0222
4	15	Intercepted the ball, Made a keypass, Made an error, Was offside, Players missing from last match, Finished on target in penalty area	−1.0091

Table 8.1: The probability assessors made by searching for best features.

Probability assessor 1 performs best. It is made using the feature search algorithm to select the best features among all. Probability assessor 2 is from the same test and differs only on “Finished with other in open play

but ball was blocked". This is a rare event and results in little difference between the two probability assessors. Unfortunately, both probability assessors use very specific categories which indicates that they could be the result of overfitting.

In order to avoid overfitting, a new probability assessor is made. Probability assessor 3 is made by taking the more general version of the categories from probability assessor 1 and 2. E.g., "Scored with right foot from set piece" was translated into "Scored". Then an exhaustive search among all subsets of these general categories is made, and the best combination turns out to be probability assessor 3.

Probability assessor 4 is based on expert knowledge. The subset is selected in [sta] where 14 different categories are selected. Then an exhaustive search for the best subset among those features is made. The score for the best subset is better than the one using only few general categories but worse than the two using very specific categories. Note that the k value is 15 which indicates that the probability assessor only needs a few neighbours for prediction.

There are now four probability assessors available: two probability assessors with very specific categories, one based on expert knowledge, and one with general categories. The union of features from these four probability assessors is used in the remainder of the settings tweaking tests.

8.4.2 Determining the Window Size

A suitable window size is determined by trying all window sizes from 1 to 12, which are the sizes which can be denoted as recent form.

The results from the test are shown in Table 8.2. Note that the score is an average over all matches which the probability assessor makes a probability assessment for. Hence, there are fewer matches for high window sizes so potentially they might avoid some easy or difficult matches.

A window size of 8 has the best logarithmic score and it uses only 13 neighbours which is only lower at window size 9. The scores generally form a curve, except for 1, 4, and 12, with window size 8 as the highest value, and hence the best score.

A window size of 4 has the second best score, but it might be due to overfitting since the features are selected to fit window size 4. However, 4 is a very attractive window size because it implies that only the first four games in the season cannot be assigned probability assessments. Furthermore, it seems to be the lowest feasible size if the scores for window sizes from 1 to 3 are considered. 4 is also the size suggested in [sta] which makes it an interesting value.

Window	k	Feature count	Log. score
1	64	10	−1.0326
2	55	5	−1.0410
3	23	11	−1.0334
4	25	9	−0.9982
5	23	9	−1.0236
6	42	9	−1.0158
7	17	12	−1.0030
8	13	9	−0.9915
9	12	9	−1.0065
10	15	14	−1.0148
11	74	4	−1.0163
12	18	9	−1.0016

Table 8.2: The best probability assessor for different window sizes.

An even window size is preferable because the teams typically play home and away alternately. This implies that a team would normally have played two home games and two away games in the last four games. A glance at statistics indicates that it is easier to play at home, as 0.457 of the games in the database ended with a home win and only 0.298 ended with an away win. Hence, an even window size would level out this difference.

The rest of the tests are performed with both window sizes 4 and 8. The reasons for this are that they perform best, they are both even, and 4 is the suggested value in [sta].

8.4.3 Determining the Normalisation

The search for the optimal normalisation is similar to the search for window size. Three types of normalisation are used: no normalisation, min-max normalisation, and Z-score. For each type, the window sizes of 4 and 8 are used in combination with the feature search algorithm to search for the best subset among the categories decided earlier. The results are shown in Table 8.3.

The best combination is the same as in the window test but the differences are very small and could be considered random. It is worth noting that window size 4 performs worse than 8 in the two new tests which might indicate that a window size of 4 is too small.

Normalisation	Window	k	Feature count	Log. score
None	4	57	14	−1.0269
None	8	43	9	−0.9920
Min-max	4	25	9	−0.9982
Min-max	8	13	9	−0.9915
Z-score	4	31	13	−1.0121
Z-score	8	25	4	−0.9997

Table 8.3: The best probability assessor for different normalisation types.

8.4.4 Settings for Tests on Fresh Data

As stated previously two different settings are to be selected and tested on previously unused data. The optimised settings, `Optimised`, are determined to be min-max normalisation, and a window size of 4 or 8. A new test, still on the learning data, is made to find the best categories in combination with these settings. The feature search algorithm is used and all features is available in the search.

A simple expert based probability assessor, `Expert`, is tested in order to indicate whether the more optimised probability assessor is overfitted.

The two selected probability assessors and their logarithmic scores, Score_1 and Score_2, are shown in Table 8.4. Score_1 is the average logarithmic score obtained on the data where the settings are determined using cross-validation, and Score_2 is the average logarithmic score on previously unused data.

This test shows that the `Optimised` probability assessor by far is the best of the two on the data where the parameters are determined. Unfortunately it performs a lot worse than the `Expert` probability assessor on yet untested data. This can only be caused by overfitting in the `Optimised` probability assessor which implies that only the `Expert` probability assessor is used for further testing.

Name	Wd.	k	Features	Score_1	Score_2
`Expert`	4	23	Made a keypass, Made an error, Scored	-1.0222	-1.0037
`Optimised`	8	42	Assisted from a set piece, Crossed to team mate on free kick, Finished off target with head in open play, Finished on target with left foot from set piece, Finished on target with other, Finished with other in open play but ball was blocked, Received a red or 2nd yellow, Scored, Scored with right foot	-0.9804	-1.0609

Table 8.4: Comparison of scores on cross-validated data and fresh data.

Chapter 9

Dixon-Coles Approach

A model is proposed in [DC97] which only considers the goals scored in the matches. The main idea is that each team has an offensive strength and a defensive weakness, α and β, which both must be above 0. A high offensive strength indicates that the team is good at scoring goals and a high defensive weakness indicates that the team is likely to concede many goals. The number of goals in a match is assumed to be based on the offensive strength and defensive weakness of the two teams. This chapter starts out by first introducing the intuition behind the Dixon-Coles approach, then it presents how the parameters and settings are determined, and lastly it shows the initial test results.

9.1 Dixon-Coles Probability Assessment

Consider two teams, where the home team has index i and the away team has index j, with offensive strengths, α_i and α_j, and defensive weaknesses, β_i and β_j. The number of home goals can be estimated by $\alpha_i\beta_j$, i.e. the offensive strength of the home team multiplied with the defensive weakness of the away team. Similarly, $\alpha_j\beta_i$ is the likely number of goals the away team scores. Note that these calculations do not take home field advantage into account which is known to be considerable. In order to do so a value, γ, is multiplied with the approximate number of home team goals, so the formula becomes $\alpha_i\beta_j\gamma$.

In order to obtain probability assessments instead of the expected counts for the number of goals scored by either of the two teams, two Poisson distributions are made with means according to the expected number of goals for each team. Poisson distributions are chosen because they resemble the way goals are distributed according to [DC97]. How they are calculated

and their properties are described in Section 4.3 on page 35. The number of home goals, if the home teams index is i and the away teams index is j, is $\boldsymbol{X}_{i,j} \sim \text{Poisson}(\alpha_i \beta_j \gamma)$ and the number of away goals is $\boldsymbol{Y}_{i,j} \sim \text{Poisson}(\alpha_j \beta_i)$ under the assumption of independence between the home and away goal counts. Rewriting this into a probability assessment for a certain number of goals for the home and away team yields:

$$\text{P}(\boldsymbol{X}_{i,j} = x,\, \boldsymbol{Y}_{i,j} = y) = \frac{\lambda^x \exp(-\lambda)}{x!} \frac{\mu^y \exp(-\mu)}{y!} \tag{9.1}$$

Where $\lambda = \alpha_i \beta_j \gamma$ and $\mu = \alpha_j \beta_i$.

According to [DC97], the independence assumption does not hold for all results, especially 0–0, 0–1, 1–0, and 1–1. In order to handle this, a factor is multiplied on the probability to compensate for the dependency. This factor, τ, is:

$$\tau_{\lambda,\mu}(x,y) \;=\; \begin{cases} 1 - \lambda\mu\rho & \text{if } x = 0 \text{ and } y = 0 \\ 1 + \lambda\rho & \text{if } x = 0 \text{ and } y = 1 \\ 1 + \mu\rho & \text{if } x = 1 \text{ and } y = 0 \\ 1 - \rho & \text{if } x = 1 \text{ and } y = 1 \\ 1 & \text{otherwise} \end{cases}$$

Note that ρ is the dependency parameter so if $\rho = 0$ it corresponds to independence between the results because $\tau_{\lambda,\mu}(x,y) = 1$ for any combination of parameters.

The dependent probability for a given result is:

$$\text{P}(\boldsymbol{X}_{i,j} = x,\, \boldsymbol{Y}_{i,j} = y) = \tau_{\lambda,\mu}(x,y) \frac{\lambda^x \exp(-\lambda)}{x!} \frac{\mu^y \exp(-\mu)}{y!} \tag{9.2}$$

In this project, the probability assessment for the outcome of a match is needed instead of the probability assessment for the number of goals scored. However, the outcome probability assessment can, easily be extracted by calculating probability assessments for the different possible results and then sum the results together based on the match outcome yielded by the score. Note that, theoretically, there are infinitely many results but most are very unlikely to happen like 9–4. In Table 9.1 selected probabilities for different results prior to the AaB–FC Midtjylland match played in the 2005–2006 season, are shown, calculated using the Dixon-Coles approach. The summations in the bottom of the table show the probability for the three possible outcomes. Note that the sum of the three different outcomes is 0.961 because some of the results are omitted, and no matter how many results are included the sum would never be 1. Because of that, it is necessary to do a normalisation of the probabilities to obtain a probability assessment.

Home win		Draw		Away win	
Score	Prob.	Score	Prob.	Score	Prob.
1–0	0.078	0–0	0.061	0–1	0.058
2–0	0.073	1–1	0.120	0–2	0.043
3–0	0.040	2–2	0.060	0–3	0.018
4–0	0.017	3–3	0.014	0–4	0.006
2–1	0.093	4–4	0.002	1–2	0.072
3–1	0.052			1–3	0.031
4–1	0.022			1–4	0.010
3–2	0.033			2–3	0.026
4–2	0.014			2–4	0.008
4–3	0.006			3–4	0.005
Sum home	0.428	Sum draw	0.257	Sum away	0.276

Table 9.1: Converting result probabilities into outcome probabilities.

9.2 Deciding the Parameters

The offensive strength and the defensive weakness of each team together with the dependency parameter, ρ, and the home field advantage, γ, need to be found in order to calculate the probabilities. First note that the values most likely change over time as a result of good and bad form, injuries, suspensions, and traded players. The model considers the values constant, hence they must be recalculated before the assessment of each match.

The values can be approximated by using a maximum-likelihood estimate. In order to prevent the model from having infinite many solutions to the equation, the following constraint is applied:

$$\frac{1}{n}\sum_{i=1}^{n}\alpha_i = 1 \tag{9.3}$$

Where n is the number of teams. This constraint results in the average offensive strength being 1. The reason why the constraint is necessary can be seen by considering some assignment of offensive strength and defensive weaknesses. If all offensive strengths are multiplied by 2 and all defensive weaknesses are divided by 2, the same probability assessments would be made, so it is impossible to decide which of them to choose.

The likelihood function is:

$$L(\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_n,\rho,\gamma) = \prod_{k=1}^{N}\tau_{\lambda_k,\mu_k}(x_k,y_k)\exp(-\lambda_k)\lambda_k^{x_k}\exp(-\mu_k)\mu_k^{y_k} \tag{9.4}$$

N is the number of matches and $\lambda_k = \alpha_{i(k)}\beta_{j(k)}\gamma$ and $\mu_k = \alpha_{j(k)}\beta_{i(k)}$ with $i(k)$ and $j(k)$ denoting respectively the indices of the home and the away team. x_k and y_k denote the number of goals scored by respectively the home and away team in match k. For the parameter estimation to make sense, all matches used in the estimation must have been played prior to the date of the match in question.

The problem with the approach in (9.4) is that each match is weighted equal. Form is considered a big factor in football [sta], so more recent matches should most likely be more dominant due to squad changes, injuries, etc.. By modifying (9.4), a pseudolikelihood at time t can be constructed as follows:

$$L(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n, \rho, \gamma) = \prod_{k=1}^{N} \left(\tau_{\lambda_k,\mu_k}(x_k, y_k) \exp(-\lambda_k)\lambda_k^{x_k} \exp(-\mu_k)\mu_k^{y_k}\right)^{\phi(t-t_k)} \quad (9.5)$$

t_k is the time where match k was played. $\phi(t)$ should yield a smaller value whenever t increases such that the most recent matches have the highest impact. This is because $\lim_{x\to 0} a^x = 1$, resulting in very old matches, with small t values, having a minimal impact because everything is multiplied together.

There are several ways to choose the ϕ function and [DC97] suggest that the following is used:

$$\phi(t) = \exp(-\xi t) \quad (9.6)$$

where ξ is a small value larger than 0. ξ needs to be determined empirically because it is not possible to optimise (9.5) with respect to ξ.

9.2.1 Optimisation of the Variable Values

The values of the variables, such as offensive strength, must be optimised by maximising the likelihood function stated in (9.5). The chosen method to find those values is gradient descent [Avr76, chap. 10].

This section brings the details about how gradient descent is implemented.

In order to perform gradient descent, it is necessary to find the partial derivatives of the function. This can be simplified by considering the log-likelihood instead of the likelihood since the logarithm function is monotonic. The log-likelihood of (9.5) is shown in (9.7).

$$
\begin{aligned}
&LL(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n, \rho, \gamma) \\
&= \ln \prod_{k=1}^{N} \left(\tau_{\lambda_k,\mu_k}(x_k, y_k) \exp(-\lambda_k) \lambda_k^{x_k} \exp(-\mu_k) \mu_k^{y_k}\right)^{\phi(t-t_k)} \\
&= \sum_{k=1}^{N} \phi(t-t_k) \ln\left(\tau_{\lambda_k,\mu_k}(x_k, y_k) \exp(-\lambda_k) \lambda_k^{x_k} \exp(-\mu_k) \mu_k^{y_k}\right) \\
&= \sum_{k=1}^{N} \phi(t-t_k) \left(\ln(\tau_{\lambda_k,\mu_k}(x_k, y_k)) - \lambda_k + x_k \ln(\lambda_k) - \mu_k + y_k \ln(\mu_k)\right)
\end{aligned}
\tag{9.7}
$$

The first task is to find the partial derivatives for all variables. Recall that $\lambda = \alpha_{i(k)}\beta_{j(k)}\gamma$ and that $\mu = \alpha_{j(k)}\beta_{i(k)}$ with $i(k)$ and $j(k)$ denoting respectively the indices of the home and the away team. First, the partial derivatives for the offensive strengths, α, are found where α_i denotes the offensive strength of the team with index i. The partial derivative of α_i is:

$$
\frac{\partial LL}{\partial \alpha_i} = \sum_{k=1}^{N} \phi(t-t_k) \begin{cases}
0 & \text{if } i(k) \neq i \text{ and } j(k) \neq i \\
\frac{-\beta_{j(k)}\gamma\mu_k\rho}{1-\lambda_k\mu_k\rho} - \beta_{j(k)}\gamma + \frac{x_k}{\alpha_{i(k)}} & \text{if } i(k) = i \text{ and } x_k = 0 \text{ and } y_k = 0 \\
\frac{\beta_{j(k)}\gamma\rho}{1+\lambda_k\rho} - \beta_{j(k)}\gamma + \frac{x_k}{\alpha_{i(k)}} & \text{if } i(k) = i \text{ and } x_k = 0 \text{ and } y_k = 1 \\
-\beta_{j(k)}\gamma + \frac{x_k}{\alpha_{i(k)}} & \text{if } i(k) = i \text{ and not handled above} \\
\frac{-\lambda_k\beta_{i(k)}\rho}{1-\lambda_k\mu_k\rho} - \beta_{i(k)} + \frac{y_k}{\alpha_{j(k)}} & \text{if } j(k) = i \text{ and } x_k = 0 \text{ and } y_k = 0 \\
\frac{\beta_{i(k)}\rho}{1+\mu_k\rho} - \beta_{i(k)} + \frac{y_k}{\alpha_{j(k)}} & \text{if } j(k) = i \text{ and } x_k = 1 \text{ and } y_k = 0 \\
-\beta_{i(k)} + \frac{y_k}{\alpha_{j(k)}} & \text{if } j(k) = i \text{ and not handled above}
\end{cases}
$$

Similarly β_i denotes the defensive weakness of the team with index i and its

partial derivative is:

$$\frac{\partial LL}{\partial \beta_i} = \sum_{k=1}^{N} \phi(t-t_k) \begin{cases} 0 & \text{if } i(k) \neq i \text{ and } j(k) \neq i \\ \frac{-\lambda_k \alpha_{j(k)} \rho}{1-\lambda_k \mu_k \rho} - \alpha_{j(k)} + \frac{y_k}{\beta_{i(k)}} & \text{if } i(k) = i \text{ and } x_k = 0 \\ & \quad \text{and } y_k = 0 \\ \frac{\alpha_{j(k)} \rho}{1+\mu_k \rho} - \alpha_{j(k)} + \frac{y_k}{\beta_{i(k)}} & \text{if } i(k) = i \text{ and } x_k = 1 \\ & \quad \text{and } y_k = 0 \\ -\alpha_{j(k)} + \frac{y_k}{\beta_{i(k)}} & \text{if } i(k) = i \text{ and not} \\ & \quad \text{handled above} \\ \frac{-\alpha_{i(k)} \gamma \mu_k \rho}{1-\lambda_k \mu_k \rho} - \alpha_{i(k)} \gamma + \frac{x_k}{\beta_{j(k)}} & \text{if } j(k) = i \text{ and } x_k = 0 \\ & \quad \text{and } y_k = 0 \\ \frac{\alpha_{i(k)} \gamma \rho}{1+\lambda_k \rho} - \alpha_{i(k)} \gamma + \frac{x_k}{\beta_{j(k)}} & \text{if } j(k) = i \text{ and } x_k = 0 \\ & \quad \text{and } y_k = 1 \\ -\alpha_{i(k)} \gamma + \frac{x_k}{\beta_{j(k)}} & \text{if } j(k) = i \text{ and not} \\ & \quad \text{handled above} \end{cases}$$

The partial derivative for γ is:

$$\frac{\partial LL}{\partial \gamma} = \sum_{k=1}^{N} \phi(t-t_k) \begin{cases} \frac{-\alpha_{i(k)} \beta_{j(k)} \mu_k \rho}{1-\lambda_k \mu_k \rho} - \alpha_{i(k)} \beta_{j(k)} + \frac{x_k}{\gamma} & \text{if } x_k = 0 \\ & \quad \text{and } y_k = 0 \\ \frac{\alpha_{i(k)} \beta_{j(k)} \rho}{1+\lambda_k \rho} - \alpha_{i(k)} \beta_{j(k)} + \frac{x_k}{\gamma} & \text{if } x_k = 0 \\ & \quad \text{and } y_k = 1 \\ -\alpha_{i(k)} \beta_{j(k)} + \frac{x_k}{\gamma} & \text{otherwise} \end{cases}$$

The partial derivative for ρ is:

$$\frac{\partial LL}{\partial \rho} = \sum_{k=1}^{N} \phi(t-t_k) \begin{cases} \frac{-\lambda_k \mu_k}{1-\lambda_k \mu_k \rho} & \text{if } x_k = 0 \text{ and } y_k = 0 \\ \frac{\lambda_k}{1+\lambda_k \rho} & \text{if } x_k = 0 \text{ and } y_k = 1 \\ \frac{\mu_k}{1+\mu_k \rho} & \text{if } x_k = 1 \text{ and } y_k = 0 \\ \frac{-1}{1-\rho} & \text{if } x_k = 1 \text{ and } y_k = 1 \\ 0 & \text{otherwise} \end{cases}$$

These derivatives can be used to find out how to alter the variable values in order to obtain a larger likelihood in the likelihood function.

In order to keep track of the current values and the partial derivatives, the ordering of the variables presented in (9.8) in vector form is used.

$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ \beta_1 \\ \vdots \\ \beta_n \\ \gamma \\ \rho \end{bmatrix} \tag{9.8}$$

It is assumed that a vector class, VECTOR, exists which is initialised with the length of the vector and zero in all entries. Note that the different indices in the vector can be accessed by $\vec{a}[i]$ which refers to the ith element in vector $\vec{a}$. The vector is 1-based so $\vec{a}[1]$ corresponds to α_1.

Recall, from (9.3), that the average offensive strength must be 1. This can be assured by projecting the descent vector onto a plane, where the average offensive strength is 1, using the surface normal to that plane. The surface normal, $\vec{n}$, to the plane is a vector where the offensive strength all are set to 1 and the defensive strength, γ, and ρ values are set to 0. The projection of the gradient vector, $\vec{grad}$, is shown in (9.9).

$$\vec{grad} \leftarrow \vec{grad} - \frac{\vec{grad} \cdot \vec{n}}{|\vec{n}|^2}\vec{n} \tag{9.9}$$

The variable value optimisation function is shown in Algorithm 9.2. The algorithm has three main phases. In lines 2–11 the default values are initialised based on the recommendations in [DC97]. The vectors have the format described in (9.8).

In lines 12–20, the gradient for the current values is found. Note that $\frac{\partial LL}{\partial \alpha_i}(\vec{values}, matches)$ implies that the $\frac{\partial LL}{\partial \alpha_i}$ function is called with the parameters in $\vec{values}$ and the matches in $matches$. The ξ and t values are expected to be set implicit in the outer environment. The normalisation of the offensive strengths is performed in line 20 which is done in order to maintain the constraint from (9.3).

In lines 21–36, the best length of the gradient vector is decided. Lines 23–26 find the longest useful length. In lines 28–35, that length is gradually minimised until the interval is shorter than 0.001. When that happens the $\vec{values}$ vector is set to a value in that interval, and the entire process of finding a new gradient starts over.

9.2.2 Determining the ξ Value

The ξ value indicates the impact of old matches on the parameter learning compared to new matches. A fitting ξ value needs to be determined for the test setup. It has to be estimated empirically independent of the other variables due to its nature in the equation which uses a similar method, as to find the best k-value, described in Section 8 on page 80. The algorithm for finding the ξ value is shown in Algorithm 9.1.

The main loop of the algorithm is lines 3–15 where the different ξ values are iterated over. If $min = 0.001$, $max = 0.004$, and $stepSize = 0.001$, the loop assures that ξ values 0.001, 0.002, 0.003, and 0.004 are tested.

Algorithm 9.1 Optimising the parameters in the likelihood function.

1: **function** OPTIMISEVARIABLES($matches, teams, precision$)
2: $\vec{values} \leftarrow$ **new** VECTOR($|teams| \cdot 2 + 2$) ▷ Make vector for values
3: $\vec{grad} \leftarrow$ **new** VECTOR($|teams| \cdot 2 + 2$) ▷ Make gradient vector
4: $\vec{n} \leftarrow$ **new** VECTOR($|teams| \cdot 2 + 2$) ▷ Vector for α normalisation
5: $\vec{last} \leftarrow$ **new** VECTOR($|teams| \cdot 2 + 2$) ▷ Vector with last values
6: **for** $i \leftarrow 1$ **to** $|teams|$ **do** ▷ For all teams
7: $\vec{values}[i] \leftarrow 1$ ▷ Initialise offensive strength
8: $\vec{values}[i + |teams|] \leftarrow 1$ ▷ Initialise defensive weakness
9: $\vec{n}[i] \leftarrow 1$ ▷ Initialise offensive strength normalisation
10: **end for**
11: $\vec{values}[2 \cdot |teams| + 1] \leftarrow 1.4$ ▷ Initialise γ

12: **while** $|\vec{last} - \vec{values}| \geq precision$ **do** ▷ Is the precision acceptable?
13: $\vec{last} \leftarrow \vec{values}$
14: **for** $i \leftarrow 1$ **to** $|teams|$ **do** ▷ For all teams
15: $\vec{grad}[i] \leftarrow \frac{\partial LL}{\partial \alpha_i}(\vec{values}, matches)$
16: $\vec{grad}[i + |teams|] \leftarrow \frac{\partial LL}{\partial \beta_i}(\vec{values}, matches)$
17: **end for**
18: $\vec{grad}[2 \cdot |teams| + 1] \leftarrow \frac{\partial LL}{\partial \gamma}(\vec{values}, matches)$
19: $\vec{grad}[2 \cdot |teams| + 2] \leftarrow \frac{\partial LL}{\partial \rho}(\vec{values}, matches)$
20: $\vec{grad} \leftarrow \vec{grad} - \frac{\vec{grad} \cdot \vec{n}}{|\vec{n}|^2}\vec{n}$ ▷ Assure that $\frac{1}{|teams|}\sum_i \alpha_i = 1$

21: $\vec{x_0} \leftarrow \vec{values}$ ▷ $\vec{x_0}$ holds first value
22: $\vec{x_1} \leftarrow \vec{values} + \vec{grad}$ ▷ $\vec{x_1}$ holds second value
23: **while** LL($\vec{x_1}$) > LL($\vec{x_0}$) **do** ▷ While better length might exist
24: $\vec{x_0} \leftarrow \vec{x_1}$
25: $\vec{x_1} \leftarrow \vec{x_1} + \vec{grad}$ ▷ Test next interval
26: **end while**
27: $\vec{x_0} \leftarrow \vec{values}$ ▷ Reset $\vec{x_0}$
28: **while** $|\vec{x_1} - \vec{x_0}| \geq 0.001$ **do** ▷ Until interval is small enough
29: $\vec{x_m} \leftarrow \frac{\vec{x_1} - \vec{x_0}}{2} + \vec{x_0}$ ▷ Find middle
30: **if** LL($\frac{\vec{x_m} - \vec{x_0}}{2} + \vec{x_0}$) > LL($\frac{\vec{x_1} - \vec{x_m}}{2} + \vec{x_m}$) **then** ▷ Best interval?
31: $\vec{x_1} \leftarrow \vec{x_m}$ ▷ First interval best
32: **else**
33: $\vec{x_0} \leftarrow \vec{x_m}$ ▷ Second interval best
34: **end if**
35: **end while**
36: $\vec{values} \leftarrow \vec{x_0}$ ▷ Update $\vec{values}$ vector
37: **end while**

38: **return** $\vec{values}$
39: **end function**

Algorithm 9.2 Find the best ξ value.

```
 1: function searchFor ξ(matches, min, max, stepSize, timePeriod)
 2:     bestξScore ← −∞

 3:     for ξ = min to max stepsize stepSize do
 4:         score ← 0
 5:         for all {m_cur ∈ matches | m_cur.time in timePeriod} do
 6:             matches_prev ← {m_prev ∈ matches | m_prev.time < m_cur.time}
 7:             assessor ← new DixonColesAssessor(matches_prev, ξ)
 8:             assessment ← assessor.assess(m_cur)
 9:             score ← score + logScore(assessment, m_cur.outcome)
10:         end for
11:         if score > bestξScore then
12:             bestξScore ← score
13:             bestξ ← ξ
14:         end if
15:     end for

16:     return bestξ
17: end function
```

The loop in lines 5–10 iterates over all matches played in the time period delivered as parameter to the function, so that the performance of the Dixon-Coles model using the current ξ values can be measured. For each match played in the time period, all matches played before that match is extracted and assigned to $matches_{prev}$. Then a Dixon-Coles assessor is made using the DixonColesAssessor class, based on the extracted previous matches using the current ξ value. When a DixonColesAssessor object is created, the OptimiseVariables function described in Algorithm 9.2 is used. This function determines the settings which are used later to make the probability assessments. That assessor is used to assess the current match in question to get a probability assessment for it. This probability assessment is evaluated using the logarithmic scoring rule, see Section 6.1.1 on page 57, based on the actual outcome of the match. The sum of the scores of all matches for the current ξ value is summed in the *score* variable.

When the scores for all matches in the time period have been assessed and the scores have been summed, the summed score is compared to the previous summed scores in lines 11–14. If the new score is better it is saved in $best\xi Score$ and the ξ value is saved in $best\xi$. When all the different ξ values have been tested, the best one is returned.

9.3 Test Results

This section presents how the test settings are determined and describes the tests which are performed.

In order to handle promoted and relegated teams, all matches where both teams currently play in the Danish SAS league or the Danish first division are included when learning the variables. The dataset used is the one specified in Section 3.3 on page 29. The matches were played during the last 6 years in the Danish SAS league and the Danish first division. There are a total of 28 teams and 4,364 matches. Only the date of each match and not the actual time of the match was considered for simplicity during calculations, so matches played earlier on the same day cannot affect a match played later.

A ξ value needs to be determined to perform the tests which can be done as described in Section 9.2.2. In [DC97], ξ values between 0 and 0.02 were tested and 0.0065 turned out as the best. All values between 0.01 and 0.02 were notable worse than 0.0065, so the search here is restricted to the span between 0.001 and 0.009 with step size 0.001. The time period for the test data is the autumn season of 2006. The results of that test is shown in Figure 9.1 and indicates that the best ξ values are in the interval between 0.004 and 0.005 with this test setup. To obtain a higher precision, the ξ values in that interval are examined with step size 0.0001 with results as shown in Figure 9.2. 0.0043 is the best value for the chosen settings, hence it is used throughout the rest of the report.

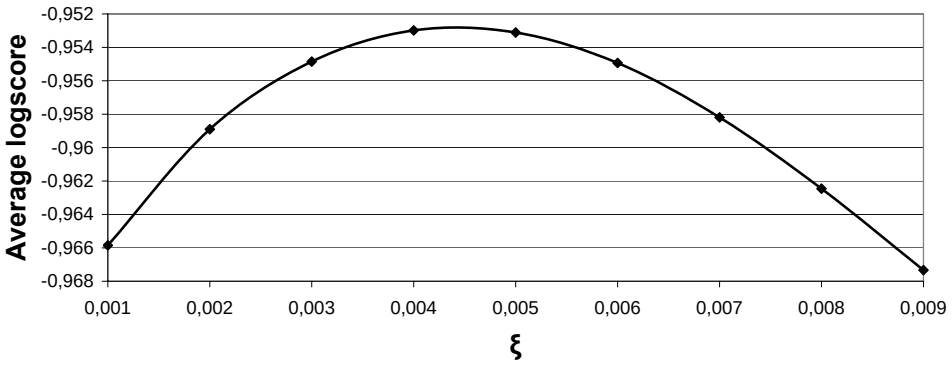

Figure 9.1: Logscores when searching for the best ξ.

9.3.1 Parameter Values

This section analyses the different parameters and argue whether they make sense. One of the advantages in the Dixon-Coles approach is that it is possible for humans to make sense of the output of the model. Table 9.2 shows

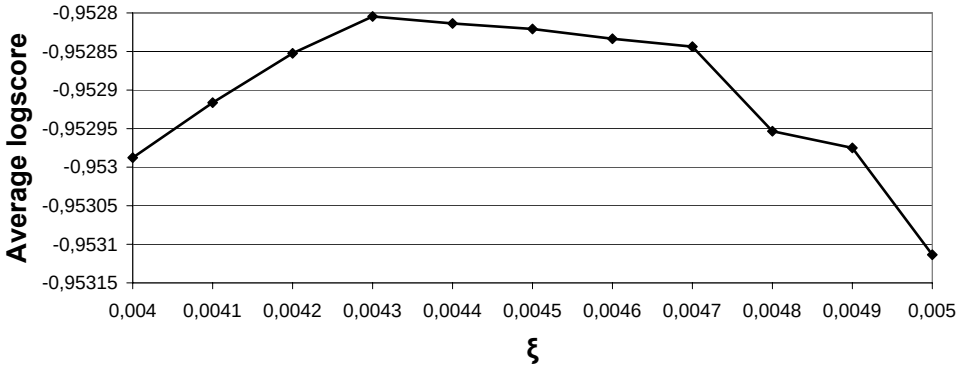

Figure 9.2: Logscores when fine-tuning the ξ value.

the 12 SAS-league teams together with their offensive strengths, defensive weaknesses, league positions, goals scored, and goals conceded. All values are from the 27th november 2007, the day after the last game of the autumn season were played.

It is clear that there exists a relation between the offensive strength and the goals scored so far this season. E.g. FC Midtjylland has scored the most goals and they have the second highest offensive strength. Vejle has conceded the most goals and has the highest defensive weakness.

Team	Offence	Defence	Pos.	Goals For	Goals Ag.
FC København	1.638	0.596	1	35	12
FC Midtjylland	1.564	0.879	2	36	20
OB	1.205	0.587	3	24	15
AaB	1.159	0.836	4	23	17
FC Nordsjælland	1.490	1.087	5	35	20
Esbjerg fB	1.353	1.121	6	31	29
Brøndby IF	1.323	0.890	7	24	21
Randers FC	1.074	1.059	8	24	26
Viborg	1.189	1.256	9	18	35
AC Horsens	0.735	0.913	10	12	23
Vejle	0.977	1.471	11	19	42
Silkeborg	0.892	1.287	12	15	36

Table 9.2: Parameters for the SAS-league teams on the 27th of November 2006.

Figure 9.3 shows how the offence of four different teams, Horsens, Silkeborg, FC København, and Brøndby, have developed over time. Round 1 in the graph corresponds to the first round in the 2005/2006 season and the line

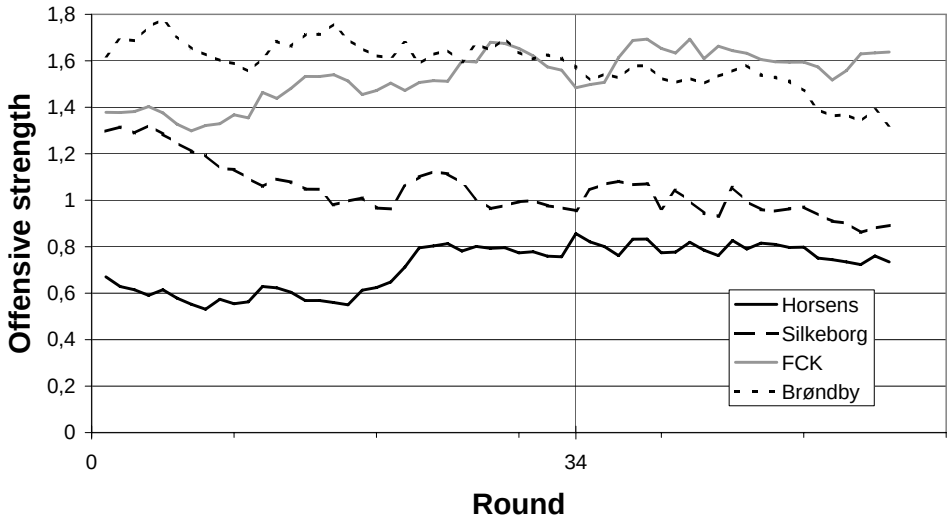

Figure 9.3: The development in offensive strengths for 1.5 seasons.

at 34 indicates the first round in the 2006/2007 season. Brøndby won the 2004/2005 league title and in the first round of the graph they are also regarded stronger than FC København. FC København performed better than Brøndby in the 2005/2006 season and the graph shows that their offensive strength gradually improves while Brøndby's deteriorates.

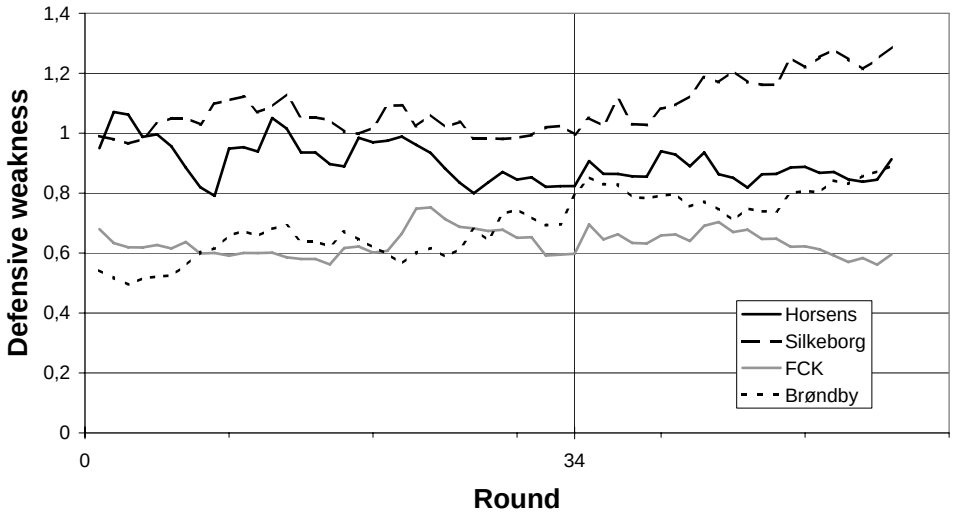

Figure 9.4: The development in defensive weaknesses for 1.5 seasons.

Figure 9.4 shows how the defences of the same four teams developed over time. Recall that a lower value is better. Note that the same pattern as in Figure 9.3 on the facing page is witnessed. Brøndby starts out better than

FC København but gradually the roles change.

Figure 9.5 shows how the home parameter, γ, developed. There are significant changes in the parameter which might be due to some teams playing better away and others playing worse home.

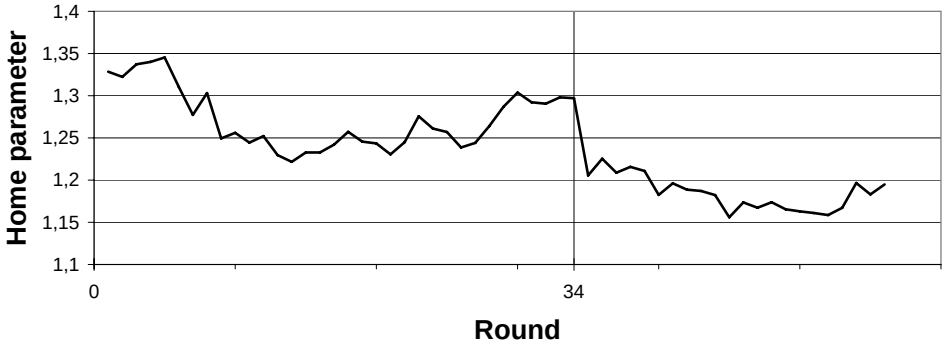

Figure 9.5: γ, the home parameter, for 1.5 seasons.

Figure 9.6 shows how the the dependency parameter, ρ, changes over time. There is no clear pattern in that graph.

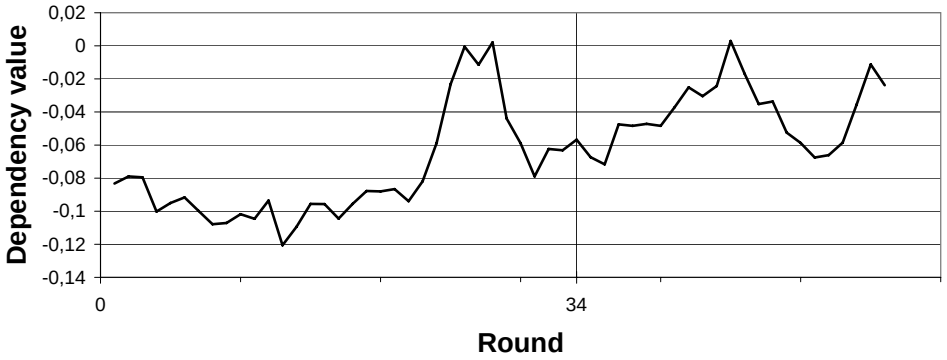

Figure 9.6: ρ, the dependency parameter, for 1.5 seasons.

9.3.2 Performance of the Dixon-Coles Approach

Using a ξ value of 0.0043 the matches in the autumn season of 2006 yield an average logarithmic score of -0.9526 and an average quadratic score of 0.4363. These scores are both very close to the available bookmakers on the same games as shown in Table 9.3. It is also worth noting that the the individual probability assessments are much closer to the bookmakers than those from the other algorithms. The Dixon-Coles model is tested further on the fresh data using a ξ value of 0.0043.

Probability Assessor	Avg. Log. Score	Avg. Quad. Score
DC	-0.9526	0.4363
NordicBet closing odds	-0.9578	0.4333
Average closing odds	-0.9523	0.4370

Table 9.3: The average scores for the Dixon-Coles approach compared to the bookmakers.

Chapter 10

Ensemble Methods

As described in Section 6.2.1, the information from the football matches is extensive, and determining which features and which algorithms to use is to some extent subjective. The performance of the decision tree algorithm os rather unstable on a dataset with a large number of features and a significant noise level, like the one used in this project. Intuitively, an aggregation of different probability assessors could make more stable probability assessments due to the different information used by each of them. Methods to combine different probability assessors are known as ensemble methods or classifier combination methods. An ensemble is created by a number of different base probability assessors which are combined into a single ensemble probability assessor using an aggregation method.

This chapter first motivates why ensemble methods can have better performance than there base probability assessors. Then different approaches for creating the base probability assessors are presented together with some implementation details. Lastly, the performance of the implemented ensembles is presented together with an analysis of the results.

Unless otherwise stated, this section is based on [TSK97], [Die00a] and [Die00b].

10.1 Motivation of Ensemble Methods

Since classification is the typical application of ensemble methods, this section starts by motivating ensemble methods from a classification point of view and then continues to consider the advantages in the probability assessment domain.

In [Die00a] ensemble methods are defined as:

Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a (weighted) vote of their predictions.

This definition says that by using some given learning algorithm, e.g. the k-Nearest Neighbor algorithm or the decision tree algorithm, it is possible to construct a number of classifications from which a voting can be performed in order to achieve an aggregated classification.

An ensemble classifier performs best when the different classifiers are diverse, implying that they to some extent indeed are different. The reason why they need to be diverse is that a classification made by an ensemble of nearly identical classifiers would yield a classification identical to the original classifications.

On the left-hand side of Figure 10.1, three different classification lines are drawn, each generated by some classification algorithm. These can be combined into an ensemble classifier with a complex decision boundary which is shown in the right-hand side of the figure as an unweighted vote between the three classifiers.

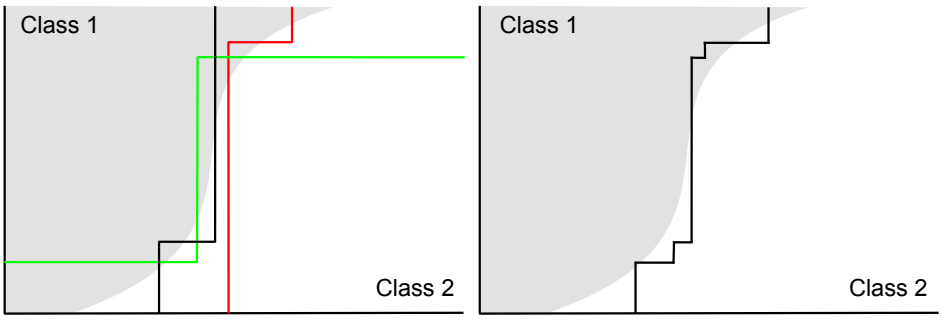

Figure 10.1: Three simple decision trees which in combination produces a complex ensemble.

Each base classifier needs to be accurate to some extent, implying that its error-rate, ϵ, must be below a given threshold. The error rate is measured by the percentage of cases classified wrongly. Each base classifier, i, has an error rate, ϵ_i, which is measured as the proportion of falsely classified instances. If a number of base classifiers with error rates above 0.5 are combined, the error rate of the ensemble classifier, $\epsilon_{ensemble}$, is worse than that of the base classifiers.

In an example with two different possible outcomes of the class variable, a number of base classifiers with a low error rate, below 0.5, would yield an ensemble classifier with $\epsilon_{ensemble} < 0.5$, assuming that the errors are independent. $\epsilon_{ensemble}$ can be calculated using (10.1) from [TSK97], where n is the number of classifiers in the ensemble. Since this is considered a

classification problem a majority vote is needed. Hence the number of votes must be more than half of n, implying that $\lceil \frac{n}{2} \rceil$ votes are needed. This can be calculated using the cumulative frequency for a binomial distribution, see [DeG89].

$$\epsilon_{ensemble} = \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \binom{n}{i} \epsilon^i (1-\epsilon)^{n-i} \tag{10.1}$$

Figure 10.2 shows $\epsilon_{ensemble}$ of an ensemble classifier as a function of 21 base classifiers, where ϵ for each of these base classifiers are identical. There are two different lines, one where the base classifiers are identical and one where the base classifiers are independent. It can be seen that it is important that the classifiers have $\epsilon < 0.5$, in order for the ensemble classifier to be better than the base classifiers. It should be noted that the more independent the base classifiers are, the better the ensemble classifier performs.

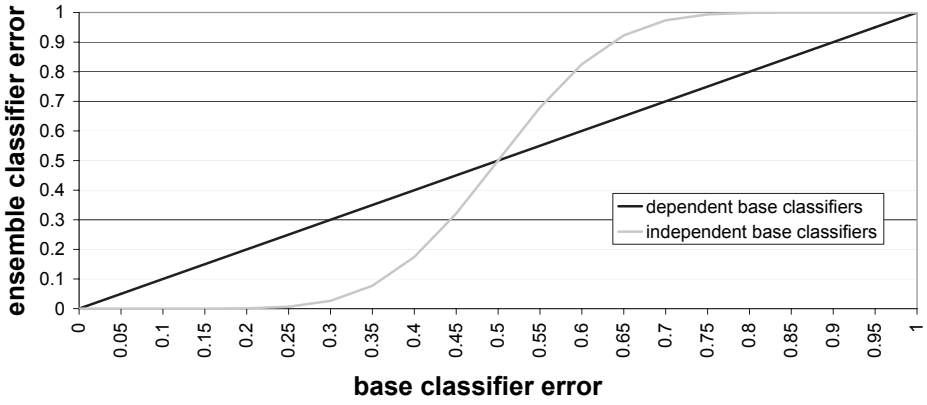

Figure 10.2: The error rate of the ensemble classifier as function of the error rate of the base classifiers.

An ensemble probability assessor uses a number of base probability assessors and combines their probability assessments into a combined probability assessment. It is desirable to prove that an ensemble made by averaging over n equally good independent base probability assessors, measured by their average logarithmic score, is at least as good as the score of the n base probability assessors on the same matches.

Before this can be proven, it must determined which methods to use for combining the base probability assessors. The easiest and most common way of combining different probability assessors is to use a simple unweighted combination, as shown in (10.2). E_o is the probability the ensemble probability assessor sets on outcome o, $Q_{o,i}$ is the probability made by the ith

base probability assessor on outcome o, and n is the number of probability assessors in the ensemble.

$$E_o = \frac{1}{n}\sum_{i=1}^{n} Q_{o,i} \tag{10.2}$$

In the following theorem, several matches are interesting but only the probability on the outcome of those matches is interesting, hence a slightly different notation is used. Let $p_{1,1}, \ldots, p_{1,m}, \ldots, p_{n,1}, \ldots, p_{n,m}$ be the probability on the outcomes made by the n base probability assessors on the m matches, such that $p_{i,j}$ refers to the probability on the outcome made by the ith probability assessor on the jth match. Then the probability on the outcome for the jth match for the ensemble is

$$p_{e,j} = \frac{\sum_{i=1}^{n} p_{i,j}}{n}.$$

Theorem 10.1 (Ensemble of multiple matches with logarithmic scoring)**.** *Let c denote the worst average logarithmic score by any of the base probability assessors, such that $\frac{\sum_{j=1}^{m} \ln(p_{i,j})}{m} \geq c$ for all i.*
Then

$$\frac{\sum_{j=1}^{m} \ln(p_{e,j})}{m} \geq c, \tag{10.3}$$

implying that the ensemble made by averaging over probability assessors bounded by a certain value scores least as well as the worst probability assessor measured by average logarithmic score.

Proof. Because of the requirements of c, the base probability assessors can be ordered in such a way that the worst is 1 making the following statement hold:

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \ln(p_{i,j})}{nm} \geq \frac{\sum_{j=1}^{m} \ln(p_{1,j})}{m} \geq c. \tag{10.4}$$

If it can be proven that $\frac{\sum_{j=1}^{m} \ln(p_{e,j})}{m} \geq \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \ln(p_{i,j})}{nm}$ it follows from (10.4) that (10.3) is true.

$$\frac{\sum_{j=1}^{m} \ln(p_{e,j})}{m} \geq \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \ln(p_{i,j})}{nm} \Leftrightarrow \tag{10.5}$$

$$\sum_{j=1}^{m} \ln(p_{e,j}) \geq \sum_{j=1}^{m} \frac{\sum_{i=1}^{n} \ln(p_{i,j})}{n} \tag{10.6}$$

Consider that proving $\sum_{i=1}^{n} a_i \geq \sum_{i=1}^{n} b_i$ can be done by proving $a_i \geq b_i$ for all $1 \leq i \leq n$. Then (10.6) can be proven by proving

$$\ln p_{e,j} \geq \frac{\sum_{i=1}^{n} \ln(p_{i,j})}{n} \tag{10.7}$$

for all $1 \leq j \leq m$. This is the same as proving that for each match, the logarithmic score for the ensemble probability assessor is at least as high as the average of the base probability assessors score. This is is stated in Theorem 10.2.

Theorem 10.2 (Ensemble of a single match with logarithmic scoring)**.** *Let $p_1, \ldots, p_n$ be the probabilities on the outcome made by n base probability assessors which have an equally good logarithmic score. Then the probability on the outcome is $p_e = \frac{\sum_{i=1}^{n} p_i}{n}$. Then define $\ln(0) = -\infty$, so that*

$$\ln(p_e) \geq \frac{\sum_{i=1}^{n} \ln(p_i)}{n} \tag{10.8}$$

This is the same as stating that an average ensemble probability assessor is at least as good at logarithmic scoring as the average of the base probability assessors.

The proof follows after Lemma 10.3.

First consider the criteria for a concave function; suppose the function f is twice differentiable on the open interval I. Then f is concave if $f''(x) < 0$ on I. For a concave function f (10.9) holds.

$$f\left(\frac{\sum_{i=1}^{n} x_i}{n}\right) \geq \frac{\sum_{i=1}^{n} f(x_i)}{n} \tag{10.9}$$

Note that the only case, in which the right hand and left hand side of the inequality are equal, is when $x_1 = \ldots = x_n$.

Lemma 10.3 (The logarithmic function is concave)**.** *The natural logarithm,* ln*, is concave on the interval $(0, \infty)$.*

Proof. The proof follows immediately from

$$\ln''(x) < 0 \Leftrightarrow \frac{1}{-x^2} < 0 \Leftrightarrow \frac{1}{x^2} > 0$$

□

Proof. (Theorem 10.2). First consider the case where $p_j = 0$ for one or more js. In that case, $\frac{\sum_{i=1}^{n} \ln(p_i)}{n} = -\infty$. Hence, the right-hand side of (10.8) cannot be larger than the right-hand side making, the theorem true.

Now the cases where $p_j > 0$ for all js are considered. In Lemma 10.3, it are proved that the logarithm function is concave, hence (10.9) holds. (10.9) can be used with $f = \ln$ and $x_1 = p_1, \ldots, x_n = p_n$, leading to (10.10).

$$\ln\left(\frac{\sum_{i=1}^{n} p_i}{n}\right) \geq \frac{\sum_{i=1}^{n} \ln(p_i)}{n} \Leftrightarrow \tag{10.10}$$

$$\ln(p_e) \geq \frac{\sum_{i=1}^{n} \ln(p_i)}{n} \tag{10.11}$$

Note that $p_e = \frac{\sum_{i=1}^{n} p_i}{n}$ as stated in (10.2). Since (10.11) and (10.8) are identical Theorem 10.2 is proved as desired. Since Theorem 10.2 holds, Theorem 10.1 also holds, hence the desired result is obtained. □

This implies that it is now known that on a sufficiently large number of matches, the logarithmic score of an ensemble probability assessor made by unweighted combination is at least as high as the scores of the worst base probability assessor, measured by their average logarithmic score. As stated above, the score of the ensemble probability assessor only the same as the worst score for the base probability assessors in cases where all base probability assessors calculates the same probability on the same match. If the base probability assessors have some degree of independence, they will often have different probabilities on the outcome, hence the ensemble probability assessor performs better than than the worst probability assessors. A large number of inequalities were used and some of them will in many cases be in favour of the ensemble, hence there is reason to believe that ensemble methods are also useful for creating a better probability assessment.

10.2 Creating the Ensembles

Given the available data, the goal is to make independent probability assessors. This is not always possible but there are several methods for obtaining at least some degree of independence. Common for these methods is that they alter the environment in which the learning is performed, such that the learning algorithm learns a number of different probability assessors [Die00b, TSK97].

- **Manipulation of the learning set, L**; in this approach L, is manipulated such that each probability assessor is created by only using a subset of L. This approach works best if the learning algorithm is unstable, implying that a small changes in L have major impact on the final probability assessor. The decision tree algorithm is an unstable learning algorithm, whereas k-Nearest Neighbor is not. Two ways of performing this approach are bagging and boosting, both are described later.

- **Manipulation of the input features;** in this approach, only a subset of the original features is used in each base probability assessor. Generally, the subset is made randomly or by a domain expert. If this approach is used, the input features should be highly redundant.

- **Manipulation of the class feature;** in cases with more than two class feature values the number of possible outcomes of the class feature, n, can be partitioned into two disjoint aggregate classes for each

classifier. These classes are denoted A_i and B_i, where i is the index of the classifier. The instances are relabelled such that those which are classified as A_i are given the derived label 0, and those which are classified as B_i are given the derived label 1. The relabelled instances are joined in a new dataset which is given to the learning algorithm in order to create a classifier h_l. k base classifiers are made, $h_1, \ldots, h_i, \ldots, h_k$, each with its own unique dataset. Each new instance, x, is classified using each h, such that if h classifies x as 0, each class in A_i receives a vote, if it classifies x as 1, each class in B_i receives a vote corresponding to its proportion of the aggregate class. This implies that if A_i contains three classes, each of these will receive $\frac{1}{3}$ vote. After each h has classified x, the class with the highest number of votes is the class-variable value of x.

- **Manipulation of the learning algorithm settings;** each base probability assessor is created with different algorithm dependent settings. In the decision tree algorithm, the minimum split size can be altered, in the k-Nearest Neighbor algorithm the number of neighbours can be changed. Moreover, there are numerous other ways of altering these algorithms.

The remainder of this section presents different well known methods which use some of these techniques.

10.2.1 Bagging

Bagging, short for bootstrap aggregation, creates a number of subsets the learning set, L [TSK97]. The learning algorithm learns the base probability assessors on these non-identical training sets, each of size n, the number of instances in the learning set. This implies that each instance can appear several times in the subsets. The instances are selected randomly, such that each instance has an equal probability of being in a each subset. In Algorithm 10.1, the subset creation process is presented. As input, it takes the learning set, L, and the number of output subsets, k.

If n and k are sufficiently large, each instance appears in roughly $\frac{2}{3}$ of the subsets. This is the case since, the probability that an instance appears in a randomly selected subset is $\lim_{n\to\infty} 1 - (1 - \frac{1}{n})^n = 1 - \frac{1}{e} \approx 0.632$.

An advantage of bagging is that it experiences less overfitting than e.g. boosting when L contains noise because an unweighted average is used. Unstable base probability assessors should be used in order to obtain the best results.

Algorithm 10.1 The bagging algorithm for selecting the subsets.

```
1: function CreateBaggingSubsets(L, k)
2:     baggingSubsets ← ∅
3:     for 1 to k do
4:         subset ← ∅
5:         while |subset| < |L| do
6:             subset ← subset ∪ L.getRandomItem()
7:         end while
8:         baggingSubsets ← baggingSubsets ∪ subset
9:     end for
10:    return baggingSubsets
11: end function
```

10.2.2 Boosting

Boosting is similar to bagging with the exception of subsets creation. Boosting is an iterative approach which adapts the subsets to the learning algorithm such that it focuses on the instances for which it is hard to make the correct probability assessments. Boosting assigns a weight to each training instance such that more difficult instances appear in more subsets, and in this way are given a higher implicit weight. Boosting is not used in this project because the complexity of the dataset would most likely result in extreme weights on some of the instances which would lead to overfitting.

10.2.3 Random Forest

Random forests generate a large number of decision trees and combine the probability assessments of these using unweighted combination. There are several methods to obtain diversity in the base probability assessors, the normal method is by changing the set of input features. It is possible to combine this method with bagging which in this project is not denoted as random forests but as bagging, in order to keep the two approaches separated.

The randomisation of the input features is done while the tree is being built. Each time a new node is created, only a limited randomly selected subset of the features is available to be split on. This subset is randomly chosen each time, such that it differs between each node. The size of this subset is in [Bre01] suggested to be either 1 or $\lceil \log_2(n) \rceil$, where n is the number of input features. The number of decision trees built is suggested to be 100, and each tree is built to its full height so that all leaves have entropy 0.

The implemented version is similar to Algorithm 7.1 on page 77. The only difference is that the original `ID3` function in line 8 uses the entire set of input features where the random forest chooses a subset of these. Additionally,

random forests create not only one decision tree but a large number of decision trees.

10.3 Initial Test Results

Three different tests are performed. The first two feature random forests with and without bagging, where bagging is used to select the instances for learning prior to creating each tree. The third test presents unweighted combinations of selected combinations of the probability assessors that performed best in previous tests.

10.3.1 Random Forests without Bagging

The settings for the tests are identical for all the different versions of random forests. The learning set, L_L, consists of 396 matches from the 04/05 and 05/06 seasons, and the test set, L_T, consists of 108 matches from the 2006 part of the 06/07 season. All features are used which is a total of 577.

The random forests approach is tried with two different number of trees; 100 and 150. The size of the features subset tried in each node is $\lceil \log_2(577) \rceil = 10$ features.

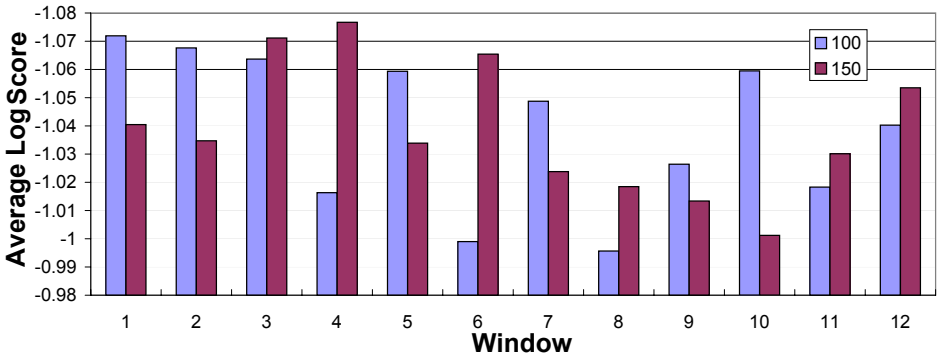

Figure 10.3: The average log scores for random forests created with 100 and 150 trees and feature subsets of size $\lceil \log_2(n) \rceil$.

Figure 10.3 shows the performance over different *window* sizes of random forests created using 100 and 150 trees and a feature subset of size $\lceil \log_2(n) \rceil$. From the figure, it can be seen that neither the random forests with a 100 trees nor the ones with 150 trees performs as well as the decision trees shown in Figure 7.1 on page 78. The best random forests are created with 100 trees and *window* sizes four, six, and eight, and 150 trees combined with *window* size ten. Eight performs best with an average log score of about -0.995.

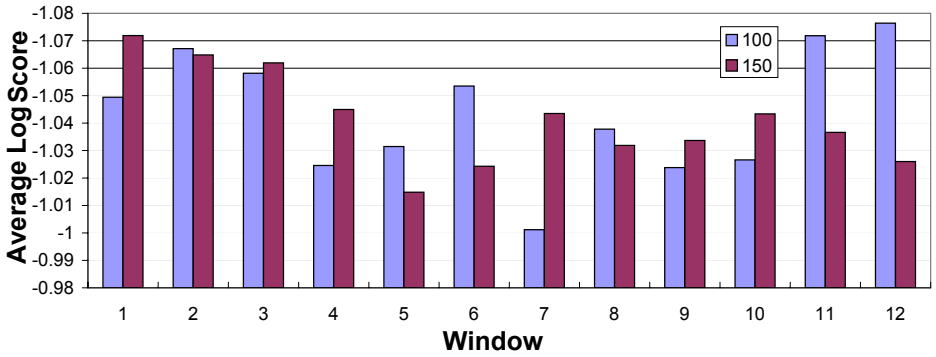

Figure 10.4: The average log scores for random forests created with 100 and 150 trees and feature subsets of size one.

Figure 10.4 shows the performance over different *window* sizes of random forests created using 100 and 150 trees and a feature subset of size one. From the figure, it can be seen that the random forests do not perform well. The best random forest is with 100 trees and a *window* size of 7 but none of the random forests have an average logarithmic score above -1.

It is ambiguous what a good *window* size is but it is clear that feature subsets of size $\lceil \log_2(n) \rceil$ are better than those of size one. This could be expected since we have a large number of features, and randomly selecting one would result in a very random result. The two best random forests using bagging in the test have 100 decision trees, the *window* sizes seven and eight, and a subset size of $\lceil \log_2(n) \rceil$. These are shown in Table 10.1.

Decision Trees	Subset Size	*Window*	Avg. Log Score
100	$\lceil \log_2(n) \rceil$	8	-0.995646
100	$\lceil \log_2(n) \rceil$	6	-0.998996

Table 10.1: The random forests with an average logarithmic score above -1.

10.3.2 Random Forests with Bagging

The settings for the test with random forests and bagging are identical to those used for random forests described in Section 10.3.1.

Figure 10.5 shows the performance over different *window* sizes of random forests created using 100 and 150 trees, bagging, and a feature subset of size $\lceil \log_2(n) \rceil$. From the figure, it can be seen that none of the random forests performs as well as the decision trees shown in Figure 7.1 on page 78. The best random forests are created with 100 trees and *window* sizes seven

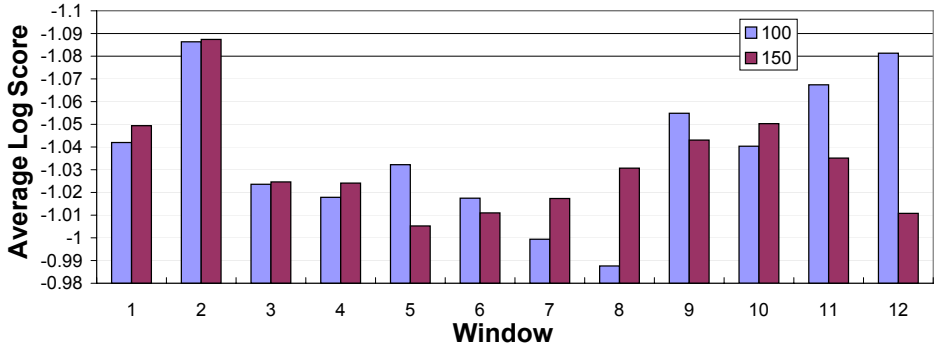

Figure 10.5: The average log scores for random forests created with 100 and 150 trees, bagging, and feature subsets of size $\lceil \log_2(n) \rceil$.

and eight. Eight performs best with an average log score of about -0.985, slightly better than the best random forest without bagging.

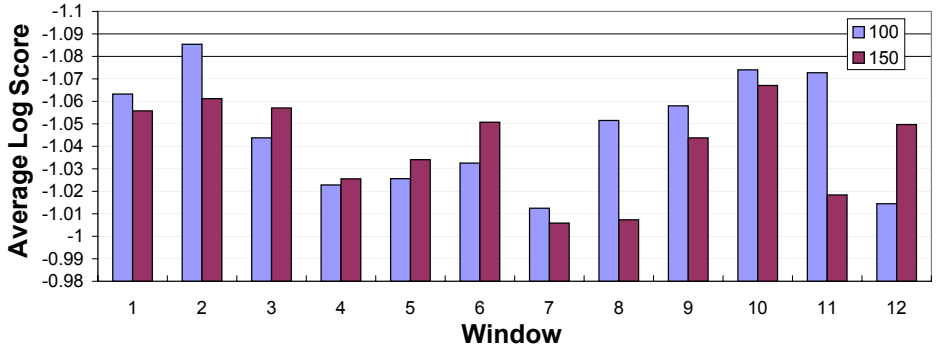

Figure 10.6: The average log scores for random forests created with 100 and 150 trees, bagging, and feature subsets of size one.

Figure 10.6 shows the performance over different *window* sizes of random forests created using 100 and 150 trees, bagging, and a feature subset of size one. From the figure it can be seen that the random forests using bagging and a feature subset of one do not perform well. None of the random forests have an average logarithmic score above -1, making them uninteresting for further exploration.

Again no clear good window size appears which was the same observation as with random forests without bagging. As can be seen in Table 10.2, only two random forest bagging probability assessors have an average logarithmic score above -1. However, the best of them is better than the best random forest probability assessor without bagging. Again, the best score comes

from the setting where the random forest is created with 100 decision trees, a feature subset of size $\lceil \log_2(n) \rceil$, and a *window* size of eight. Hence this seems to be the optimal setting for random forests, and bagging seems to slightly improve the logarithmic score for this setting.

Decision Trees	Subset Size	*Window*	Avg. Log Score
100	$\lceil \log_2(n) \rceil$	8	-0.987623
100	$\lceil \log_2(n) \rceil$	7	-0.999379

Table 10.2: The random forests using bagging with an average logarithmic score above -1.

10.3.3 Unweighted Ensembles of Probability Assessors

The third test is to shows how an ensemble of previously created probability assessors performs. This is done with an unweighted combination using some of the top decision trees from Table 7.1 on page 78, the `Expert` k-Nearest Neighbor probability assessor, the two random forest probability assessor from Table 10.1, the two random forest probability assessor created using bagging from Table 10.2, the Dixon-Coles probability assessor, and the average probabilities from 50 bookmakers. This approach is, for the decision trees, effectively randomisation, since the settings of the learning algorithm are altered, in this case both the *minsize* and the *window* value. Note that the number of matches varies according to the *window* size. The eight combinations with the best test results can be seen in Table 10.3.

The table shows that the ensemble probability assessor that uses the best decision tree together with the Dixon-Coles probability assessor performs extremely well. The result is far better than the individual logarithmic scores of the two probability assessors. The structures of the two learning algorithms are very different, implying that the two base probability assessors are more diverse than two probability assessors created with the same algorithm. This fits with the description in Section 10.1, where it was stated that the more diverse the base probability assessors are, the more significant the improvement to the ensemble probability assessor is.

The ensemble created based on three of the top decision trees, also performs extremely well along with one created by two decision trees with the Dixon-Coles probability assessor. The other ensemble probability assessors do not perform as well, though they still perform better than the base probability assessors from which they are made. Note that the logarithmic scores are calculated on the same data as the original probability assessors were tested on. This implies that only probability assessors known to be good were used in the ensembles, so another test on fresh data should be made.

Id	Matches	Learning Algorithm	Settings	Avg. Log Score	Avg. Ensemble Log Score
1	76	Dixon-Coles	Standard	−0.95081	−0.93082
		Decision Tree	*window*=4, *minsize*=28	−0.96550	
2	46	Decision Tree	*window*=4, *minsize*=28	−0.96656	−0.93897
		Decision Tree	*window*=6, *minsize*=40	−0.96164	
		Decision Tree	*window*=9, *minsize*=24	−0.98514	
3	64	Dixon-Coles	Standard	−0.96121	−0.93990
		Decision Tree	*window*=4, *minsize*=28	−0.97303	
		Decision Tree	*window*=6, *minsize*=40	−0.98240	
4	76	Dixon-Coles	Standard	−0.95713	−0.94085
		Decision Tree	*window*=4, *minsize*=28	−0.96550	
		bookmaker	average from 50 bookmakers	−0.94085	
5	52	Dixon-Coles	Standard	−0.95081	−0.94626
		Decision Tree	*window*=4, *minsize*=28	−0.96864	
		random forests	*window*=8, Decision Trees=100, bagging	−0.98762	
6	76	Dixon-Coles	Standard	−0.96774	−0.94663
		Decision Tree	*window*=4, *minsize*=28	−0.96550	
		k-Nearest Neighbor	*window*=4, *neighbours*=23	−0.99747	
7	52	Dixon-Coles	Standard	−0.95081	−0.94993
		Decision Tree	*window*=4, *minsize*=28	−0.96864	
		random forests	*window*=8, Decision Trees=100	−0.99564	
8	58	Decision Tree	*window*=4, *minsize*=28	−0.99124	−0.95706
		Decision Tree	*window*=6, *minsize*=40	−0.96860	
		Decision Tree	*window*=7, *minsize*=80	−0.96929	

Table 10.3: The best ensembles created using unweighted combination of previously created probability assessors.

An additional thing to notice from the table is that a combination of different *window* sizes appear to work well. Ensemble 2 is a combination of three different *window* sizes, and this ensemble performs better than the two other ensembles, 4 and 5, with only decision trees. Ensemble 2 has a lower sample size, number of matches, and it appears to be easier to make probability assessments for these according to the average logarithmic scores but this could still be an area to investigate further. Lastly, it can be seen that both the ensemble probability assessor which uses random forests as well as the one which uses the bookmaker, do not perform well compared to those who make it to the table.

Chapter 11

Test Results

This chapter starts by presenting the probability assessors and the settings used in the final test which are the best ones found in the previous chapters. Furthermore, details of which tests are performed are described. The different probability assessors are evaluated as bookmakers first and gamblers afterwards. Finally, the different approaches for probability assessor creation are compared based on pros and cons together with the test results.

11.1 Test Plan for Final Evaluation

In Chapters 7 to 10, four different methods for making probability assessments were presented and one approach for combining them. During the selection process, overfitting might have happened, hence a test on the validation set only using the top probability assessor found for each learning algorithm is performed. This section describes the settings chosen for each method and the tests that are performed for each probability assessor.

11.1.1 Learning Algorithms and Settings

In Table 7.1 on page 78, a number of well performing decision trees were presented. Since a tree is a statically learned model there are two possible ways of testing the best settings; either the original tree can be reused or alternatively a new tree can be learned on the learning and test data used in that section. The advantage obtained by using the same tree is that it might fit the problem domain well partly due to luck. On the other hand, a newly learned tree on more data uses more learning instances and hence under normal circumstances would be considered better. Since both approaches have advantages, they are both tested. The original tree used

the features "Overall Made a keypass", "Overall Finished with left foot in open play shot was blocked" and, "Overall Crossed to team mate on corner". It had a *minsize* of 28 and a *window* size of 4. The average logarithmic score obtained with these settings was -0.9655.

The settings for the k-Nearest Neighbor algorithm were found in Section 8.4.4 on page 89. The k value was 23, the window size 4, and min-max normalisation was used. The features were "Made a keypass", "Scored", and "Made an error". The average logarithmic score was -1.0222 which is somewhat worse than the bookmakers.

The Dixon-Coles approach provided the best result both alone and in combination with other learning algorithms. The stand alone average logarithmic score was -0.9526. Dixon-Coles has only one setting, ξ, which decides the weighting between old and new matches. The best ξ in the learning data was 0.0043.

Ensemble methods were used to combine different base probability assessors into more powerful combinations using two general strategies. Random forests using at least 100 base probability assessors were build, but the best average logarithmic score was -0.9876 based on a window size of 8 with bagging so it does not seem a useful method for this problem domain. Another approach used some of the previous probability assessors as a base and combined them to obtain better results. The best one was based on the decision tree described above in combination with Dixon-Coles so this combination is tested. The average logarithmic score was -0.9308 for this combination which is better than the available bookmakers on the same matches.

Furthermore, some bookmakers are considered in the tests, these were presented in Section 3.4 on page 31 as NordicBet's closing odds and the average closing odds, denoted Avg. BM odds.

All the probability assessors which are used in the final test are shown in Table 11.1.

Prob. assessor	Short name	Avg. log score	Variance
Dixon-Coles	`Dixon-Coles`	-0.9318 ± 0.3425	0.0200
Dixon-Coles + DT.	`DC + Old DT`	-0.9365 ± 0.3491	0.0207
Avg. BM odds	`BM Avg.`	-0.9831 ± 0.4103	0.0242
Random Forest	`Random Forest`	-0.9876 ± 0.3518	0.0177
NordicBet	`NB`	-0.9940 ± 0.4345	0.0258
Decision Tree	`Old DT`	-1.0060 ± 0.4636	0.0276
k-Nearest Neighbor	`kNN`	-1.0222 ± 0.2864	0.0128

Table 11.1: Scores for the used probability assessors in the tuning phase.

11.1.2 Tests to Be Performed

All learning and tuning of the probability assessors was performed on matches played prior to the 1st of January 2007, and all of the following tests are performed on matches played after the 1st of January 2007, as described in Section 6.3.2 on page 69. This ensures that the learning, tuning and validation is performed on disjoint datasets which provides the most accurate test results, as described in Section 5.4 on page 51.

In Section 1.2 on page 3, the primary goal was stated as finding a probability assessor suitable for decision support for bookmakers and the secondary goal as finding a probability assessor which is useful for gambling.

The different selected probability assessors was first evaluated for bookmaking. This is done by comparing their logarithmic and quadratic scores with those from bookmakers. The comparison also includes both a dependent t-test and a Wilcoxon Signed-Rank Test to determine whether the differences are statistically significant. The required level of significance, the α value, is set to 0.05. After that the Bookmaker-Gambler scoring rule is applied to each of the probability assessors against NordicBets closing odds. The most interesting good and bad results are explored in more detail in the process, including histograms of the probability assessments made by some of the probability assessors.

The evaluation as a gambler is performed by calculating the expected gain obtained by placing a stake at the different outcomes. If any of the expected gains are above a pre-set threshold value the stake is simulated as placed. The wins and losses calculated based on the actual outcomes are then summed and it can be determined what the return of investment would have been.

11.2 Results as Bookmaker

To make a correct evaluation of the performance for bookmaking all the models should participate in the odds market. This is, however, not possible. Hence, some other way of comparison must be found. What is done is that both the average logarithmic and quadratic scores for all probability assessors and bookmakers are calculated together with statistical significance levels for differences in the logarithmic scores. The Bookmaker-Gambler score is also calculated between each bookmaker versus each probability assessor. In this manner many different tests are performed and it is assumed that if these test results all point in the same direction, then a probable conclusion can be made.

The average logarithmic and quadratic score on the validation set is shown in Table 11.2, together with their standard deviation and the variance of

the probability assessments. The table shows that both bookmakers outperform every automatically made probability assessor, both measured by the logarithmic and the quadratic scoring rules. It is expected that both the standard deviation and the variance decrease when ensembles are used and even the simple ensemble of two probability assessors, `DC + Old DT`, confirms this.

Prob. assessor	Avg. log score	Avg. quad score	Variance
`NB`	-0.9607 ± 0.4330	0.4326 ± 0.3014	0.0293
`BM Avg.`	-0.9631 ± 0.3926	0.4301 ± 0.2774	0.0247
`Dixon-Coles`	-0.9849 ± 0.4535	0.4163 ± 0.3152	0.0302
`DC + Old DT`	-1.016 ± 0.3979	0.3954 ± 0.2714	0.0217
`Random Forest`	-1.083 ± 0.3025	0.3453 ± 0.2084	0.0109
`kNN`	-1.085 ± 0.3884	0.3427 ± 0.2670	0.0183
`Old DT`	-1.090 ± 0.4834	0.3475 ± 0.3200	0.0267
`New DT`	-1.125 ± 0.6030	0.3232 ± 0.4109	0.0426

Table 11.2: Test of the probability assessors on the validation set.

The poor results can partly be explained by a combination of several factors. First, some overfitting occurred when the probability assessors were selected because several thousand probability assessors were made but only a few became part of the final test. Hence, the selected probability assessors were slightly overfitted with regard to the test set used in the learning phase. Furthermore, this spring season has been very surprising as teams which won numerous games in the autumn season suddenly have started losing many games, or losing teams have started winning. This is the case for at least four teams.

The bookmakers have a natural edge over the probability assessors in the first rounds, as there have been training matches, injuries, and trades prior to the first game of the spring season which the probability assessors are unaware of. Table 11.3 shows the average scores for the test set where the first two rounds of the spring season are removed. It is clear that all probability assessors improve significantly, also the bookmakers, implying that the surprising results also had some impact on the performance of the bookmakers. Though every probability assessor improves, `Dixon-Coles` is the one which improves the most, and it nearly reaches the same level as `BM Avg`

11.2.1 Significance Levels

Statistical significance tests were described in Section 4.5 on page 37 and they can be used to determine whether the differences in the logarithmic

Prob. assessor	Avg. log score	Avg. quad score
NB	−0.9048	0.4747
BM Avg.	−0.9118	0.4686
Dixon-Coles	−0.9203	0.4638
DC + Old DT	−0.9689	0.4299
Random Forest	−1.0615	0.3596
kNN	−1.035	0.3759
Old DT	−1.0592	0.3674
New DT	−1.0909	0.3444

Table 11.3: Test of the probability assessors on the validation set without the two first rounds.

scores occurred by chance or whether they can be expected to occur again in another experiment, that is, on future matches.

Both the dependent t-test and the Wilcoxon signed-rank test are used here in order to determine which probability assessors are significantly better than other, measured by their logarithmic score. Note that the bookmakers were found to be better than the created probability assessors, hence the test of statistical significance is primarily focused on which probability assessors the bookmakers are better than.

The t-test assumes normal distributions. In the case of the dependent t-test, the differences between the pairs must be normally distributed and in the case of the independent t-test, the sample must be normally distributed. To illustrate why the independent t-test cannot be used consider Figure 11.1. This figure shows the distribution of probabilities on the actual outcome made by NordicBet on the validation set. The distribution peaks in the interval from 0.2–0.3 and a again from 0.4–0.5 and 0.6–0.7. These peaks indicates that the probabilities are not normally distributed. The combined columns are grouped by outcome in Figure 11.2. The first peak is caused by a high number of draws, predicted with a probability of 0.2–0.3, whilst the second is a combination of away wins and home wins predicted with a probability of 0.4–0.5. Note that NordicBet never predicts draw with a higher probability than 0.3.

In Figure 11.3, the distribution of probabilities on the actual outcome made by the Dixon-Coles probability assessor on the matches in the validation set is shown. It is clear that it is not a normal distribution due to the two separate tops in the intervals 0.2–0.3 and 0.4–0.5. Note further that the overall distribution of probabilities is very similar to NordicBets distribution. The explanation for the two peaks can be seen in Figure 11.4; the first peak is caused by a high number of draws, whilst the second is caused by a combination of away wins and home wins.

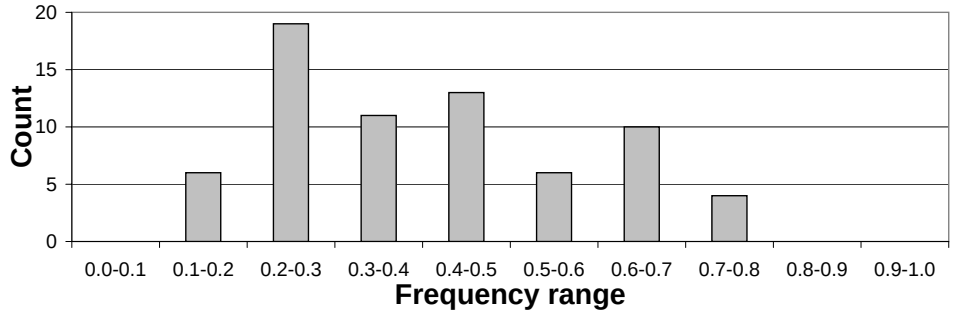

Figure 11.1: Histogram for the combined probabilities on all the actual outcomes from NordicBet.

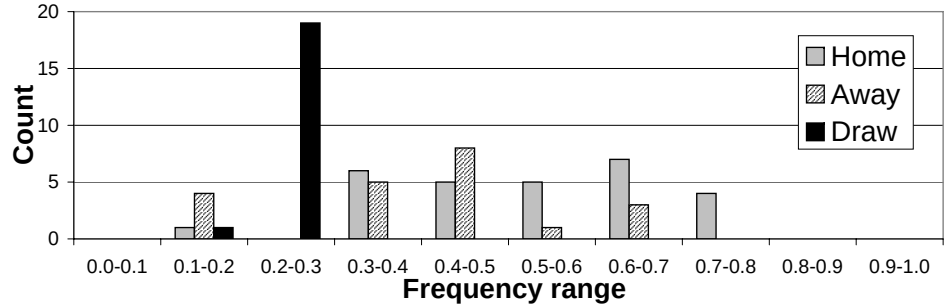

Figure 11.2: Histogram for the probabilities on actual outcomes from NordicBet.

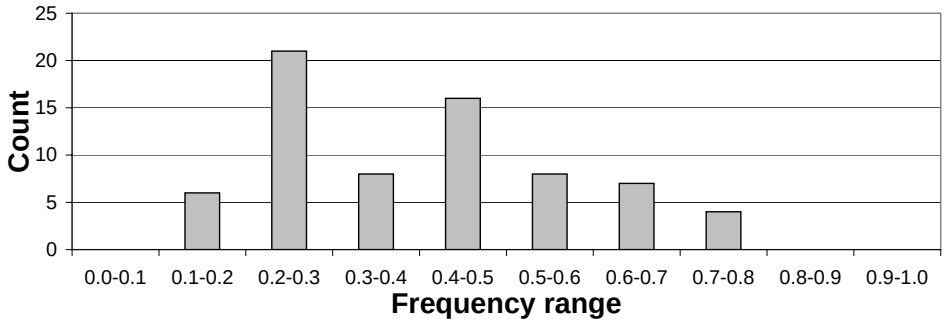

Figure 11.3: Histogram for the combined probabilities on all the actual outcomes made by the Dixon-Coles approach.

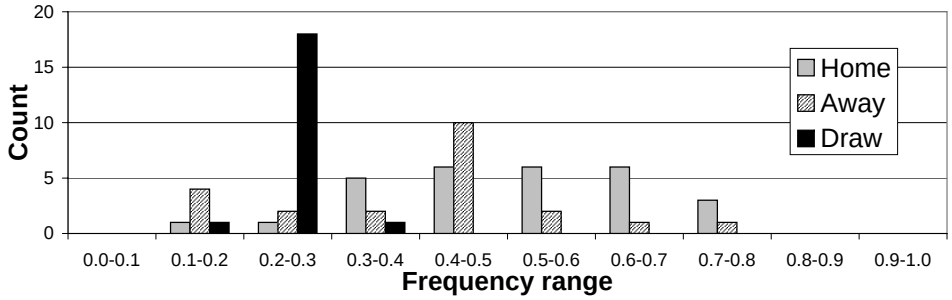

Figure 11.4: Histogram for the probabilities on the actual outcome made by the Dixon-Coles approach.

Neither of the graphs appear to be normally distributed and the first peak is caused by the large number of draws predicted. Actually, the Dixon-Coles never predicts draw with a probability larger than 0.4. Figure 11.5 shows the probability for draw using Dixon-Coles and it is clear that the area where probabilities are above 0.4 is very limited. The highest probabilities occur whenever the two teams are about equally strong and when μ and λ are close to 0, implying that neither of the teams have a high chance of scoring any goals.

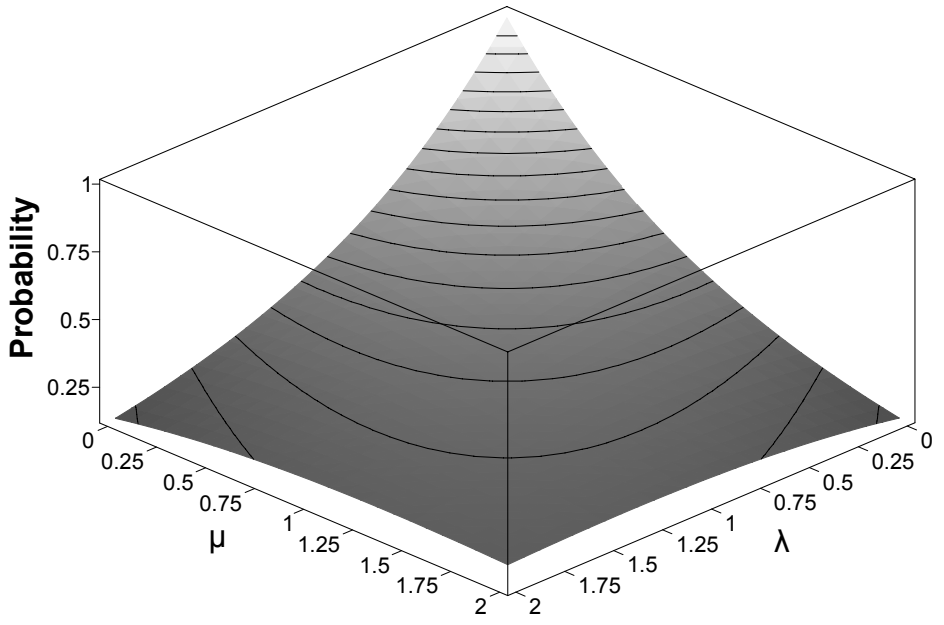

Figure 11.5: Dixon-Coles Probability on draw given $\lambda = 0$, γ, and μ.

The peak around 0.4–0.5 in Figure 11.3 stems from predictions on away wins. This is also reflected in Figure 11.6 which shows that as λ decreases, and μ increases, the probability for away win increases to close to 1. Generally,

the probability for away win is large, and roughly 0.5 of the area have an away win probability of 0.4 or more. Note, however, that λ is larger than μ on average because the home team factor is a part of λ.

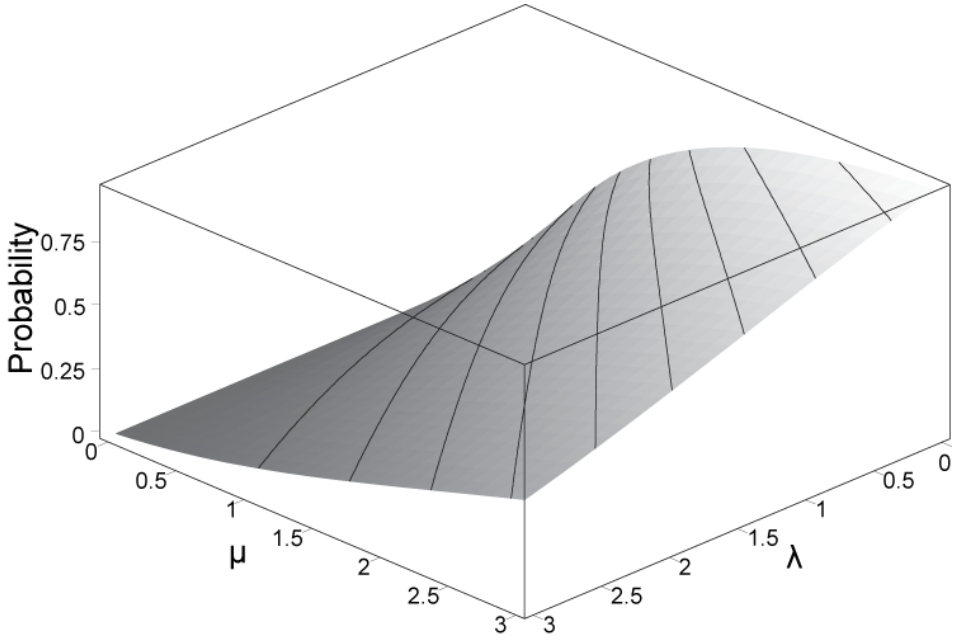

Figure 11.6: Dixon-Coles Probability on away win given $\lambda = 0$, γ, and μ.

Because the samples are clearly not normally distributed, independent t-test is not an option. Dependent t-test, described in Section 4.5.4 on page 38 requires that the differences between the pairs fit the normal distribution. This only is true for some of the combinations of probability assessors, more details regarding that will be presented later.

Hypothesis test with both the Wilcoxon-Signed Rank Test and the dependent t-test was applied to the probability assessors. Table 11.4 shows the p-values for the tuning test and Table 11.5 shows the p-values for the validation test. All values written in boldface are statistical significant at the 0.05 level. Note that the dependent t-test is used regardless of the differences between the pairs are not normally distributed.

The upper number in each grid in the table shows the p-value for the t-test. The null hypothesis is that the normal distribution of the pairs have mean 0 or lower, the alternative hypothesis is that mean is higher than 0. The normal distribution is made by for each common match the logarithmic score from the probability assessor in the column header is subtracted from the logarithmic score of the probability assessor in the row header. This implies that if the p-value is below 0.05 the null hypothesis can be rejected and the average logarithmic score of the probability assessor in the row header is

	DC+DT	BM Avg.	RF	NB	DT	kNN
DC	0.5863 0.2359	**0.0148** **0.0003**	0.2090 0.0534	**0.0125** **0.0010**	0.3115 0.4301	0.0723 **0.0026**
DC+DT		0.0638 0.2249	0.0621 **0.0371**	**0.0398** 0.1950	0.1395 0.5289	**0.0102** **0.0056**
BM Avg.			0.4573 0.2679	**0.0294** 0.9202	0.5357 0.7102	0.2110 **0.0280**
RF				0.4436 0.6557	0.6488 0.8504	0.6043 0.3681
NB					0.6197 0.7362	0.3276 **0.0308**
DT						0.1940 **0.0137**

Table 11.4: Hypothesis tests on probability assessors evaluated on the tuning dataset, with t-test on the top and the Wilcoxon Signed-Rank Test on the bottom.

significantly better than the probability assessor in the column header.

The Shapiro-Wilk test for normality were only performed on the test in the validation set. The null hypothesis for the Shapiro-Wilk test is that the sample came from a normally distributed population and the alternative hypothesis is that it did not. In Table 11.5 the p-values from the t-test are underlined if the Shapiro-Wilk null hypothesis holds with less than 0.05 chance, implying that underlined values are not reliable.

The lower number in each grid in the table shows the p-value for the Wilcoxon-Signed Rank Test. The null hypothesis is that the median of the pairs is 0 or lower, the alternative hypothesis is that median is higher than 0. The differences between the pairs is made by for each common match the logarithmic score from the probability assessor in the column header is subtracted from the logarithmic score of the probability assessor in the row header. This implies that if the p-value is below 0.05 the null hypothesis can be rejected and the average logarithmic score of the probability assessor in the row header is significantly better than the probability assessor in the column header.

Table 11.4 shows that DC is significantly better than both bookmakers, both measured by the dependent t-test and the Wilcoxon Signed-Rank Test in the test on the tuning dataset. Furthermore DC+Old DT is significantly better than NB using the dependent t-test. It is clear that kNN and RF do not perform well whereas DT performs mediocre with only a single significant result, where it is better than kNN using the Wilcoxon Signed-Rank Test.

Table 11.5 shows that both bookmakers are significantly better than most of

	BM Avg.	DC	DC+Old DT	RF	kNN	Old DT	New DT
NB	0.9705 0.0798	0.1831 0.3660	0.0605 **0.0055**	**0.0011** **0.0000**	**0.0019** **0.0001**	**0.0078** **0.0010**	**0.0053** **0.0315**
BM Avg.		0.0960 0.3258	**0.0403** **0.0151**	**0.0003** **0.0000**	**0.0007** **0.0002**	**0.0076** **0.0019**	**0.0061** 0.0535
DC			0.1238 **0.0116**	**0.0093** **0.0003**	**0.0073** **0.0004**	**0.0149** **0.0018**	**0.0073** **0.0460**
DC+Old DT				**0.0210** **0.0005**	**0.0257** **0.0020**	**0.0011** **0.0011**	**0.0037** 0.1332
RF					0.4702 0.6972	0.4344 0.9285	0.2364 0.8851
kNN						0.4569 0.8451	0.2514 0.8242
Old DT							0.1318 **0.0052**

Table 11.5: Hypothesis tests on probability assessors evaluated on the validation dataset, with t-test on the top and the Wilcoxon Signed-Rank Test on the bottom.

the automatically made probability assessors, both measured by the dependent t-test and the Wilcoxon Signed-Rank Test on the validation dataset. Only the pure DC probability assessor is not beaten by the bookmakers at an 0.05 significance level. Using the dependent t-test, the combination of DC+Old DT, is not significantly worse but it is very close, hence it is considered to be worse.

If the significance test of the probability assessors on the validation dataset is compared to that on the tuning dataset, it should be clear that the results are noteworthy different. Both NB and BM Avg. are close to being significantly better than DC in the new test whereas DC is better than the bookmakers in the test on the tuning dataset. kNN is significantly worse than most other probability assessors in both tests. Furthermore, the test on the tuning dataset showed that neither of the bookmakers were significantly better than any of the probability assessors with exception of kNN but the new tests contradict this by showing that both bookmakers are significantly better than most other probability assessors with exception of DC. As described in Section 4.5.3 on page 38, the fact that a result is statistically significant could have been caused by an error, in the sense that an α-value of 0.05, as used in these tests, imply a 0.05 probability of the result being erroneous.

11.2.2 The Bookmaker-Gambler Scoring Rule

The probability assessors are also tested using the Bookmaker-Gambler scoring rule, described in Section 6.1.2 on page 60. Table 11.6 shows the average score of the probability assessors versus both NordicBet and the average odds, respectively.

Name	NordicBet Closing	Avg. Odds
`NB`	0.0	−0.2148
`BM Avg.`	0.2148	0.0
`DC`	−0.0342	−0.4215
`DC + Old DT`	−0.5421	−0.3639
`RF`	−0.8530	−0.9083
`kNN`	−1.0308	−1.0120
`Old DT`	−1.0870	−1.0739
`New DT`	−0.9317	−1.0552

Table 11.6: The performance of the probability assessors measured by the Bookmaker-Gambler scoring rule.

The table shows that all probability assessors lose to the bookmakers, and that `BM Avg.` is the best bookmaker. `DC` is very close to NordicBet but does not perform as well against the average odds. The ensemble, `DC + Old DT`, performs slightly worse than `DC`, whereas all the other probability assessors perform about equally bad.

The results are similar to those obtained with the logarithmic and quadratic scoring rules which indicates it makes sense to apply the Bookmaker-Gambler scoring rule to this problem domain.

11.3 Results as Gambler

As described in Section 2.2 on page 6, the gambler can calculate his expected gain for a given bet if he knows the odds and he has made a probability assessment on the game. The probability assessment can be made by a probability assessor, hence the probability assessor can be evaluated for gambling use.

The main idea is to calculate the expected gain on each possible outcome for each match. Note that the odds to choose are the highest available on the given outcome on the given match, because it is possible to place the bet at any bookmaker. If the expected gain is above a given threshold, a bet is placed otherwise betting is avoided. The size of the bet can either be determined by the size of the expected gain or be fixed to the same value on

each bet. When the bet is placed, the actual payback can be calculated based on the outcome of the match. When the gain for each game is calculated, the gains for all matches are summed up, such that it is possible to see how much would have been lost or won if the scheme was actually used in real life on all the matches.

In this test, a stake of 1 unit on each bet is chosen. The results are shown in Table 11.7. Note that only the `DC` and the `DC + Old DT` probability assessors are tested, as these are the only two not determined to be significantly worse than the bookmakers. A threshold value of 0.2 signifies that the gain is expected to be at least 0.2 of the stake. Stake is the total stake placed, won is the amount paid back, and profit is won minus stake. R.O.I. is the return of investment, i.e. how much the gambler gets in return per unit invested. Note that it is possible to play on more than one outcome of a match.

Name	Stake	Won	Profit	R.O.I	Profit matches
Threshold 0.0					
`NB`	21	9.25	-11.75	-0.5595	4
`BM Avg.`	35	41.14	6.14	0.1757	9
`DC`	76	65.14	-10.85	-0.1428	22
`DC + Old DT`	88	62.68	-25.31	-0.2877	18
Threshold 0.1					
`NB`	1	0.0	-1.0	-1.0	0
`BM Avg.`	7	14.15	7.15	1.0214	2
`DC`	34	21.30	-12.69	-0.3735	7
`DC + Old DT`	50	36.80	-13.19	-0.2639	9
Threshold 0.2					
`NB`	0	0.0	0.0	0.0	0
`BM Avg.`	3	0.0	-3.0	-1.0	0
`DC`	15	5.09	-9.90	-0.6600	2
`DC + Old DT`	33	30.00	-2.99	-0.0909	6
Threshold 0.3					
`NB`	0	0.0	0.0	0.0	0
`BM Avg.`	1	0.0	-1.0	-1.0	0
`DC`	7	0.0	-7.0	-1.0	0
`DC + Old DT`	24	24.90	0.90	0.0375	4

Table 11.7: The performance of the probability assessors as gamblers.

The table shows that `DC + Old DT` has positive return of investment when the threshold is 0.3 but only four bets of 24 made were won. Hence, the profit could occur by chance. Furthermore, some of the losses obtained by both `DC` and `DC + Old DT` are large, indicating that they are not useful for

gambling.

An interesting observation is that the `BM Avg.` probability assessor has a positive return of investment with both threshold 0.0 and 0.1. Even though the profit is obtained by playing only on few matches, it still indicates that it might be possible to beat the bookmakers only using odds from other bookmakers. `BM Avg.` is made by averaging over 150 bookmakers which is closely related to ensemble methods and also to the approach in [DMP+06] where the assessments from a number of experts are combined to yield very good assessments. [PW02] presents a similar results where they are able to beat the bookmakers by using the market average in spread betting, a betting form where the pay-off is based on the precision of the prediction instead of the fixed win or loss considered in this report.

11.4 Evaluation of the Different Approaches

This section evaluates the different approaches with regard to strength and weaknesses. The different types of probability assessors have different strengths and weaknesses, as shown in schematic form in Table 11.8. Note that this is a discussion of the actual models constructed in this project, not of the general application of the algorithms on other datasets or in other domains.

	`kNN`	`DT`	`Dixon-Coles`
Makes probability assessments	×	×	×
Fine grained probability assessments			×
Data freely available			×
Luck in previous matches	×	×	
Handles promotion/relegation			×
Player absence	(×)	(×)	
Handles cross division/country			×
Assessments similar to bookmaker			×
Handles game importance	(×)	(×)	

Table 11.8: The strengths and weaknesses of the different types of probability assessors.

Common for all types is that they make probability assessments which was a prior requirement stated in the project goals.

11.4.1 Grain Size

One factor which could cause both the bookmaker and the gambler results to appear worse is that the grain of the probability assessment could be

too rough. That is, the probability assessment can only have finitely many values. One example is that the k value in k-Nearest Neighbor determines how fine-grained the probability assessment of that k-Nearest Neighbor is, since a small k value only allows few different probability assessment levels. This effect only occurs in some of the probability assessors of this test, the bookmakers are free to select the probability assessment they choose and the Dixon-Coles approach also has an infinitely fine grain, as well as the ensembles it is part of.

The grain size of both the k-Nearest Neighbor algorithm and the decision tree algorithm can be calculated. For k-Nearest Neighbor the possible probabilities are $0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k}$. The implementation of `kNN` uses $k = 23$, hence the probability assessments can be made with an interval of $\frac{1}{23} \approx 0.0434$. E.g. if the true probability for an outcome is 0.1, `kNN` can only assess 0.0868 or 0.1302, as these are the closest. If `kNN` had been able to predict within the entire interval, the maximum logarithmic score is -1.0181 which is considered the upper bound for the score.

This effect is even more noteworthy for `Old DT`, shown in Figure 7.3 on page 79. As shown in Table 11.9, the various intervals between the possible outcomes are of different size, some very large, and there are only seven different sets to choose from. E.g. if the true probability for a match is (0.35, 0.2, 0.45), the closest probability that `Old DT` can assign is (0.452, 0.226, 0.323). This is not particularly close. It yields a much better score if the outcome is home win, a slightly better score if the outcome is a draw, and a much worse score if the outcome is an away win. The upper bound for `Old DT` on logarithmic score is -0.9653. The possible probabilities for `New DT` are similar though there are only four possible sets of probabilities and the upper bound for the logarithmic score is -0.8750. Note that the upper bound is not an expression of a realistically obtainable score due to the low number of sets, hence the intervals are very large.

Home Win	Draw	Away Win
0.775	0.025	0.200
0.700	0.225	0.075
0.611	0.083	0.306
0.474	0.192	0.333
0.452	0.226	0.323
0.267	0.356	0.378
0.182	0.286	0.533

Table 11.9: The possible probabilities for `Old DT`.

The overall effect of the probability assessments having too large intervals between their possible probabilities is that both wins and losses tend to get

larger, such that scores are more extreme that they should be. Due to the usage of proper scoring rules it also always yields an expected lower score to change the probability assessment.

11.4.2 Dataset Issues

The advantage of the Dixon-Coles approach is that it only uses the number of goals scored in the model, hence the dataset is freely available on the internet for almost all leagues on all levels for the entire world. This implies that most difficulties which arise from promotion and relegation can be avoided, and that the strength of leagues in different countries can be compared to each other.

The two other models use more complex datasets constructed by Statman, hence the data is difficult to obtain. Furthermore, due to the complexity of the dataset, it is very hard to deal with promotions because no data is available for the promoted teams.

The Dixon-Coles probability assessor also has a built in ability to determine how good a team is compared to a team in another league or country which is not the case for the other two approaches.

11.4.3 Feature Issues

Both k-Nearest Neighbor and decision tree can handle matches where the previous results were effected by luck or coincidence because they can use features like key passes and shots on goal. They can also partly handle player absence if features which handles this are created. Furthermore, they can partly handle importance of matches if a feature could be constructed which handles this.

The main disadvantage with Dixon-Coles is that it only uses the number of goals scored from past matches. The number of goals scored is typically very low so luck or coincidence has an influence on the total number. Furthermore, the importance of the match and information about the upcoming match available prior to the match, e.g. the line-up, are unused by this approach.

11.4.4 Probability Assessors Roundup

Even though the different approaches have different strengths, the results were pretty clear: Dixon-Coles was the only approach which performed close to the bookmakers and in the end that is by far the most important criteria. It should be clear that no perfect solution exists but that if a model can

be constructed which combines the positive traits for the different types of probability assessors, that is a good solution.

Unfortunately, the tests in the previous sections, where the probability assessors were tested as both bookmakers and gamblers, showed that none of probability assessors constructed from neither the k-Nearest Neighbor algorithm nor the decision tree algorithm could compare with the bookmakers. Though the ensemble constructed from the Dixon-Coles and the decision tree approach performed better than the base decision tree probability assessor, it did not perform better than the base Dixon-Coles probability assessor. This was caused by the significant difference between the two base models, and even though they were combined the ensemble could not improve the score of the base decision tree enough.

In hindsight, this could be caused by numerous decision trees being learned on the learning set, L_L, including the one used in the final validation. These were all tested in the test set, L_T, and the best settings were selected before the final test. This approach is likely to have overfitted the learned model with respect to L_T. The appropriate approach would have been to first determine the settings using cross-validation on L_L, then select a few of the best settings, and only test those on L_T such that a much smaller number of probability assessors would have been tested on L_T, hence decreasing the chance of overfitting. This approach was used in k-Nearest Neighbor where the overfitted model was spotted and hence a more simple model was used in the final test.

Chapter 12

Conclusion

This chapter evaluates the project against the goals stated in Section 1.2 on page 3 and suggests how better results can be obtained.

12.1 Evaluation of Project Goals

The main goal of the project was to establish a model which can output probability assessments on outcomes of football matches and which performs so well that it can be useful for a bookmaker. The initial tests indicated that a combination of the Dixon-Coles approach and decision trees would result in such a model.

Unfortunately, the test on fresh data indicated that the prior result were caused by overfitting, hence the goal was not fully achievable since no better scores for any of the probability assessors in the tests were obtained. To verify the result, the dependent t-test and the Wilcoxon Signed-Rank Test were applied to the logarithmic scores, and the results showed that only Dixon-Coles are no worse than the bookmakers at a 0.05 significance level. Dixon-Coles is the only probability assessor with a performance nearly as good as the bookmakers, and an ensemble between Dixon-Coles and a probability assessor based on more feature types or an alteration of it might be what is needed to achieve improvements.

The results were the same when the different probability assessors were evaluated as gamblers. However, Dixon-Coles did beat the bookmakers at one specific threshold value. This result unfortunately seems to be by chance rather than general. The results may have suffered from too few matches because if stakes only are placed on 15 or fewer bets, small differences in match outcomes can lead to big differences in profits.

The goals stated that the probability assessors should be built on data available before the match being played. This goal was achieved by using historic data, and the experiments indicated that using data from the last four matches yields the best results in most cases. The two key events which were used by most of the created probability assessors were deviations of keypasses and finishes.

In order to determine the accuracy of the different probability assessors, two absolute and a pair-wise scoring rule, the Bookmaker-Gambler scoring rule, were used. The Bookmaker-Gambler scoring rule was developed in this project as a domain specific scoring rule which fits the problem domain.

12.2 Suggestions for Improvements

As described in the evaluation of the project goals, Dixon-Coles is nearly as good as the bookmakers. This section proposes some initiatives which may improve the Dixon-Coles approach.

The most obvious one is to combine Dixon-Coles with one or more probability assessors using ensemble methods. The initial tests indicated that a probability assessor based on decision trees in combination with Dixon-Coles was better than the bookmakers but the particular decision tree performed bad on the fresh data, hence the ensemble also performed bad. If a decision tree or k-Nearest Neighbor setting can be determined which yields good performance in general, this might be the key for improvement of Dixon-Coles.

The reason, why an ensemble of Dixon-Coles and another type of probability assessor is believed to be better than the individual probability assessors, is that Dixon-Coles has problems with previous matched that were decided by luck or absent players. This is caused by only the number of goals scored being considered when the offensive strengths and defensive weaknesses are calculated. If these parameters were influenced by other factors like e.g. keypasses and absent players, the parameters might be more accurate, hence the probability assessments would also be more accurate.

More data is also desirable, both odds from more bookmakers and detailed data from more matches. When evaluating the models for gambling more odds would allow the model to play at bookmakers with higher odds which will lead to higher profits.

More detailed match data would allow more levels in the decision trees which would result in smaller grain sizes and hence a higher level of flexibility in the probability assessments. As shown in Section 11.4 on page 131 the upper bound for the average logarithmic scores for the current decision trees are better than the score of the bookmakers, implying that it should be possible to improve the scores of the current decision trees.

Bibliography

[Abd06] H. Abdi. Z-scores. *Encyclopedia of Measurement and Statistics*, 2006.

[Avr76] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Prentice Hall, 1976.

[Bre01] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[CDLR02] M. Crowder, M. Dixon, A. Ledford and M. Robinson. Dynamic modelling and prediction of English Football League matches for betting. *Journal of the Royal Statistical Society: Series D (The Statistican)*, 51(2):157–168, 2002.

[CH07] Tobias Christensen and Rasmus Dencker Hansen. Odds assessment on football matches, January 2007.

[DC97] M.J. Dixon and S.G. Coles. 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, 1997.

[DeG89] M.H. DeGroot. *Probability and Statistics*. Addison-Wesley, 2. edition, 1989.

[DHS01] R.O. Duda, P.E. Hart and D.G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., 2.. edition, 2001.

[Die00a] T.G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.

[Die00b] T.G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857, 2000.

[DMP+06] V. Dani, O. Madani, D. Pennock, S. Sanghai and B. Galebach. An empirical comparison of algorithms for aggregating expert predictions. *UAI*, 2006.

[Fli02] K. Flinders. Football Injuries are Rocket Science. *vnunet.com*, Oct 2002.

[Geo07] B. Geoghegan. Gaming for the Gaming Generation. *Hospitality Upgrade*, (1):142–143, 2007.

[Geu02] Pierre Geurts. Contributions to decision tree induction: bias/-variance trade-off and time series classification, May 2002.

[GR04] T. Gneiting and A.E. Raftery. Strictly proper scoring rules, prediction and estimation. 2004.

[HK01] J. Han and M. Kamber. *Data Mining – Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[HMS01] D. Hand, H. Mannila and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.

[JFN06] A. Joseph, NE Fenton and M. Neil. Predicting football results using Bayesian nets and other machine learning techniques. *Knowledge-Based Systems*, 19(7):544–553, 2006.

[JI03] Goddard J. and Asimakopoulos I. Modelling football match results and the efficiency of fixed-odds betting. 2003.

[Kee95] E.S. Keeping. *Introduction to Statistical Inference*. Dover, 1995.

[Lar05] D.T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. John-Wiley and Sons. Inc., 2005.

[Lew03] M. Lewis. *Moneyball - The Art of Winning an Unfair Game*. W. W. Noton & Company, inc., 2003.

[Mit97] T.M. Mitchell. *Machine learning*. McGraw-Hill, 1997.

[nor] Mail correspondence with Nikolaj Juul Nielsen and Frederik Søvang from NordicBet, 2006.

[PW02] D. Paton and L.V. Williams. Quarbs and Efficiency in Spread Betting: can you beat the bookie? 2002.

[RS00] H. Rue and O. Salvesen. Prediction and Retrospective Analysis of Soccer Matches in a League. *Journal of the Royal Statistical Society: Series D (The Statistican)*, 49(3):399–418, 2000.

[sta] Mail correspondence with Kresten Buch from Statman ApS, 2006.

[SW65] S.S. Shapiro and M.B. Wilk. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika*, 52(3/4):591–611, 1965.

[TSK97] P.N. Tan, M. Steinbach and V. Kumar. *Introduction to Data Mining*. Pearson Education, Inc., 1997.

[Wil45] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[WM68] R.L. Winkler and A.H. Murphy. Good probability assessors. *Journal of Applied Meteorology*, 7(4):751–758, 1968.

[WM97] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions*, 1(1):67–82, 1997.

[Zwi03] D. Zwillinger. *CRC Standard Mathematical Tables and Formulae*. CRC Press, 31. edition, 2003.

Appendix A

Categories

Category	k-Nearest Neighbor	Decision Tree
Assisted		
Assisted from a set piece	×	
Assisted in open play		
Assisted on a cross		
Assisted on corner		
Blocked attempt		
Blocked free kick attempt		
Blocked the ball		
Cleared by head		
Cleared the ball		
Committed a misconduct		
Completed defensive long pass		
Completed defensive short pass		
Completed dribbling		
Completed mid zone long pass		
Completed mid zone short pass		
Completed offensive long pass		
Completed offensive short pass		×

Table A.1: The Statman categories, A–C

Category	k-Nearest Neighbor	Decision Tree
Completed pass		
Crossed to opponent		
Crossed to opponent from set piece		
Crossed to opponent in open play		
Crossed to opponent on corner		
Crossed to opponent on free kick		
Crossed to team mate		
Crossed to team mate form set piece		
Crossed to team mate in open play		
Crossed to team mate on corner		×
Crossed to team mate on free kick		
Finished off target		×
Finished off target with foot		
Finished off target with head		
Finished off target with head in open play		
Finished off target with left foot from set piece		
Finished off target with left foot in open play		
Finished off target with other in open play		
Finished off target with right foot from set piece		
Finished off target with right foot in open play		
Finished on target		
Finished on target on free kick		
Finished on target with foot		
Finished on target with head		
Finished on target with head in open play		
Finished on target with left foot		
Finished on target with left foot from set piece		
Finished on target with left foot in open play		
Finished on target with other		
Finished on target with other in open play		
Finished on target with right foot		
Finished on target with right foot from set piece		
Finished on target with right foot in open play		
Finished on woodwork		
Finished with head in open play, blocked		
Finished with head but ball was blocked	×	

Table A.2: The Statman categories, C–F

Category	k-Nearest Neighbor	Decision Tree
Finished with left foot from set piece, blocked		
Finished with left foot in open play, blocked		×
Finished with other in open play but ball was blocked	×	
Finished with right foot from set piece shot was blocked	×	
Finished with right foot in open play, blocked		
Flicked the ball		
Free kick shot off target	×	
Intercepted the ball	×	
Keypassed from set piece	×	
Keypassed in open play		
Keypassed on a cross		
Keypassed or assisted on a cross		
Lost a blocking		
Lost an interception		
Made a keypass	×	×
Made a tackle		
Made an error	×	
Missed a penalty		
Received 2nd yellow card		
Received a red or 2nd yellow	×	
Received a yellow or 2nd yellow		
Received red card		
Received yellow card		
Saved and deflected out of play		
Saved and deflected to opponent		
Saved and deflected to team mate		
Saved and held the ball		
Scored	×	
Scored an own goal		
Scored in open play by other		
Scored on direct free kick	×	
Scored on penalty		
Scored with head		
Scored with head in open play		
Scored with left foot		
Scored with left foot from set piece		

Table A.3: The Statman categories, F–S

Category	k-Nearest Neighbor	Decision Tree
Scored with left foot in open play		
Scored with other		
Scored with right foot		
Scored with right foot from set piece	×	
Scored with right foot in open play		
Tackled to opponent		
Tackled to team mate		
Took a corner		×
Took a goal kick		
Took a throw in		
Uncompleted defensive long pass		
Uncompleted defensive short pass		
Uncompleted dribbling		
Uncompleted mid zone long pass		
Uncompleted mid zone short pass		
Uncompleted offensive long pass		×
Uncompleted offensive short pass		
Uncompleted pass		
Was offside	×	
Was substituted		
Won a blocking		
Won an interception		

Table A.4: The Statman categories, S–W

Category	k-Nearest Neighbor	Decision Tree
Finished off target in the goal area		
Finished off target in the penalty area except the goal area		
Finished off target outside the penalty area		
Finished on target in the goal area	×	
Finished on target in the penalty area except the goal area	×	
Finished on target outside the penalty area		
Players missing from last match	×	

Table A.5: The constructed categories