Predicting Football Outcomes

with Bayesian Networks

by

Max van Dijk

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Thursday July 11, 2019 at 15:30.

Student number:	4584252	
Project duration:	April 23, 2019 – July 11, 2019	
Thesis committee:	Dr. ir. G. F. Nane,	TU Delft, supervisor
	Drs. E. M. van Elderen,	TU Delft
	Dr. ir. L. J. J. van Iersel,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Voetbal wedstrijden voorspellen

met Bayesiaanse Netwerken

door

Max van Dijk

ter verkrijging van de graad van Bachelor of Science
aan de Technische Universiteit Delft,
in het openbaar te verdedigen op donderdag 11 juli om 15:30.

Student nummer:	4584252	
Project duur:	23 april 2019 – 11 juli 2019	
Thesis commissie:	Dr. ir. G. F. Nane,	TU Delft, begeleider
	Drs. E. M. van Elderen,	TU Delft
	Dr. ir. L. J. J. van Iersel,	TU Delft

TU Delft

Abstract

In this thesis Bayesian Networks are used to predict European football matches between the years 2008 and 2016. The goal of this research is to see how the structures learned by different Bayesian Network learning algorithms influences the predictions. First the data is explored and modified to be used for Bayesian Networks and secondly the theory is explained using examples. Finally the theory is applied on the data and the structures are learned with the help of a bootstrap method and the predictions are validated using 5-fold cross validation. We can conclude that the networks learned by the algorithms and with the help of an expert give a good representation of the underlying relationships, but are not very good in prediction the end result.

Contents

Abstract ii

1 Introduction 1

2 Football Data Analysis 2
2.1 Data modification . . . 3
2.1.1 Best betting company . . . 5
2.2 Recap . . . 7

3 Bayesian Networks 9
3.1 Probability theory . . . 9
3.2 Graph theory . . . 10
3.3 Types of Bayesian Networks . . . 12
3.4 Structure Learning . . . 12
3.4.1 Constraint based learning algorithms . . . 13
3.4.2 Score based learning algorithms . . . 13
3.4.3 Hybrid learning algorithms . . . 14
3.4.4 Experts . . . 14
3.5 Parameter fitting . . . 14

4 The Bayesian Network Football Structure 15
4.1 Bootstrap . . . 18
4.1.1 Predicting . . . 18
4.2 K-fold Cross Validation . . . 21
4.3 Combining bootstrap graph with cross validation . . . 22
4.4 Expert knowledge . . . 23
4.4.1 Bootstrap . . . 23
4.4.2 K-fold cross validation . . . 25
4.4.3 Combining bootstrap graph with cross validation . . . 27

5 Conclusion 29

A Table betting companies 30

B Additional Strength plots 31
B.1 Simple bootstrap . . . 31
B.2 Bootstrap with all arcs towards goals . . . 33
B.3 Bootstrap without betting nodes . . . 34

C Code 35
C.1 Python code . . . 35
C.1.1 Functions . . . 35
C.1.2 Importing . . . 41
C.2 R code . . . 42
C.2.1 Best better . . . 42
C.2.2 Functions . . . 43
C.2.3 Main . . . 44

Bibliography 47

1 Introduction

Football is the most popular sport in the world. It is played by around 3 billion people world wide, mainly in South-, Central America and Europe. The main tournament is the World Cup which is played every four years in a different country. Besides that every continent has its own Championship and almost all countries have their own football league as well. Because there are so many people watching and playing this sport, a lot of money is involved with predicting the match outcome. That is why in this report we would like to improve those betting predictions with the help of Bayesian Networks.

Bayesian Networks give a graphical representation of a causality model. This is also our first assumption as of course correlation does not always imply causation. Each node in the graph represents a variable and each arc represents a causality relation. A second assumption that has to be made, is that there are no latent variables, so no unobserved variables that might influence the variables in the structure. As for the football game there are a lot of variables to consider, which also makes it interesting to bet on, but on the other hand makes it difficult to predict. The main advantage of Bayesian Networks is that they give a simple graphical representation of a complex underlying probability distribution.

The data used for this research is obtained from the database website `kaggle.com`. This data set contains around 25 000 European football matches between 2008 and 2016 and per match attributes like the goals for the home and away team are given.

The question we want to answer is whether we can predict these matches with the help of Bayesian Networks, and if we can how well did we do? The data set also contains betting predictions of multiple betting companies, so we would also like to know if the predictions made by the Bayesian Networks are better than those of the betting company.

In chapter 2 the data is closely analyzed and some modifications are made to make the data ready to use with Bayesian Networks. Next, in chapter 3 the theory of Bayesian Networks is explained with the help of a simple example. We also consider in this chapter the algorithms that are to be used to create our network structures. In the fourth chapter the algorithms are applied to the data and validation methods are implemented to prove the validity of our structures. Next predictions are being made and tested against the real data. Lastly some modifications to the network are done to hopefully improve the prediction results.

2

Football Data Analysis

In this section we will take a closer look at the data available. First the correlations are computed for some of the variables. Thereafter the data is modified to include more correlated variables. Lastly, we select the betting company which predicts matches the best.

We obtained the data from the database website kaggle.com. This website contains thousands of data sets which can be used to perform analysis on. Our soccer database consists of seven separate databases. We have to link a couple of these data sets to perform our analysis on. The most important data set for us is the one with all the match results. This set contains around 25 000 matches from multiple countries in Europe. The games were played in the period between 2008 and 2016. Each match has 115 attributes, such as date, home goals, away goals and stage. Furthermore, the betting odds of nine different betting companies are given. These odds are split in three columns. So we have separate columns for the odds for a home win, an away win and a draw. An example is given in table 2.1. The betting works as follows: if for this match you place a bet on a win for the away team, and it is a win for the away team then you get 5 times your bet back, but if it was a draw or a win for the home team you lose your bet.

Betting company	B365H	B365D	B365A
Betting odd	1.73	3.40	5.00

Table 2.1: Betting odds for a random match

Besides a data set with match details there is also a set with player statistics. These statistics are obtained from sofifa.com, which gets the statistics from the FIFA football game. From these player statistics we will only use the player's overall rating. These overall ratings are determined by the game publisher EA. Some details about how they determine these ratings can be found here[1]. As can be seen in the interview these ratings can be a bit arbitrary but it's the best statistic we've got for the players.

The last data set we will use consists of team statistics. These statistics are again obtained from sofifa.com. From this data set we will only use the build up play speed, because it describes the rating of the team the best as possible, independent of which players start the game. The play speed can be divided into three classes, slow, balanced and fast. While this play speed is mainly used in the computer game to determine how fast your team has to move forward to start an attack, it can also be determined by looking at what the teams are doing inside the field. So for example if a team is very defensive it will have a low build up play speed, while if a team always tries to move the ball to the front as fast as possible, it will have a high build up play speed.

Each team, player, country, league and match gets a number assigned which is used as an

[1] https://global.espn.com/football/blog/espn%2Dfc%2Dunited/68/post/2959703/fifa%D17%2Dplayer%2Dratings%2Dsystem%2Dblends%2Dadvanced%2Dstats%2Dand%2Dsubjective%2Dscouting

identification number. With the other available data sets we can convert these numbers into the name that number represents. So for example the team number 8593 represents Ajax and 8634 represents FC Barcelona. Similarly the number 30981 represents the world famous player Lionel Messi. With these ID numbers it is very easy to figure out what rating belongs to which player because in both data sets this number is given and it can also be used to figure out which players start at the beginning of the match.

2.1. Data modification

The first thing we do is compute the result of the match, this means that we need to determine which team won the game. To do that we have to establish first which teams scored the most goals. If the home team won we give the match a label "2", if it was a draw we give the match label "1" and if it the away team won, we give it label "0". Besides this we calculate what the goal difference is. To make use of Bayesian Networks we need variables which describe a cause and effect relationship. To model this we make use of the correlation between variables. This is determined with the Pearson Correlation, given by equation 2.1, for random variables X and Y with expectation μ_X and μ_Y and variance σ_X and σ_Y respectively.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{2.1}$$

If we compute this correlation for some variables in the set we get the correlation plot in figure 2.1. The stage of a match represents the round of the national championship and the day number is the day of the week, given by number 1 for Monday and number 7 for Sunday. If the circles are dark blue, the variables are strongly correlated and if the circles are dark red the variables are strongly negatively correlated. However as we can see, most of the matrix is white, which implies that these variables are very weakly correlated with one another. There are a few things that stand out though, firstly the *match_api_id* is very strongly correlated with the *year*, this means we can make a linear function between these two variables. This makes sense because every match gets a unique id number and this increases with the years. Secondly *home_team_goal* is slightly correlated with the betting prediction, which will be useful to predict the goals with the betting prediction.

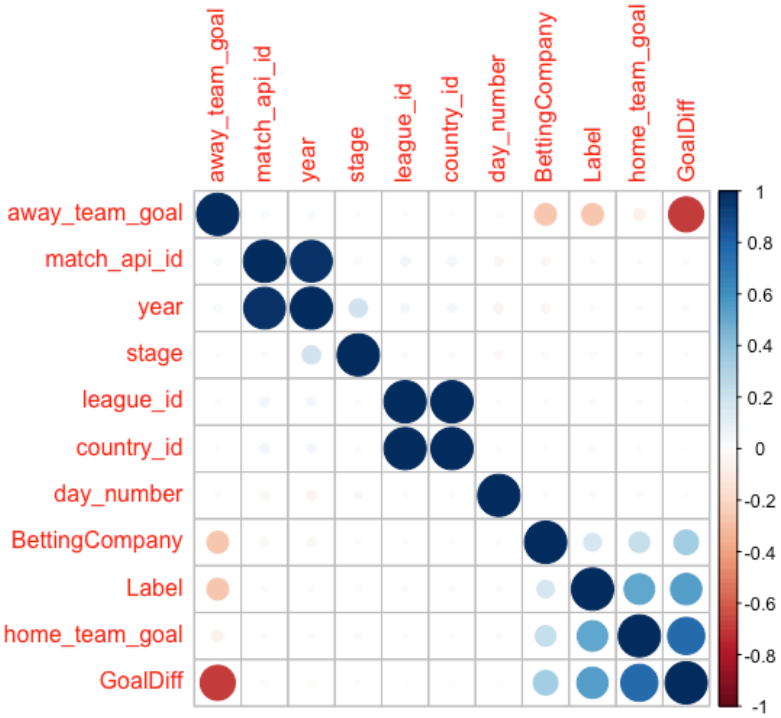


Figure 2.1: Correlation plot for some variables

Due to the fact that most of the correlation matrix is white, we will have to create some new columns with more relevant data. For each of these columns we will make a histogram and the corresponding fitted normal distribution, as we want to see if our assumption holds that the data follows a normal distribution. The first thing we will look at is the player rating. However, the ratings are only determined once every year so we will have to assume that these ratings stay constant until there is a new update. Now we look up which players are starting in the match and what their current rating is. Next we take the average of the players to get the mean overall rating. For a visualization of the ratings we make a histogram plot. This plot can be seen in figure 2.2.

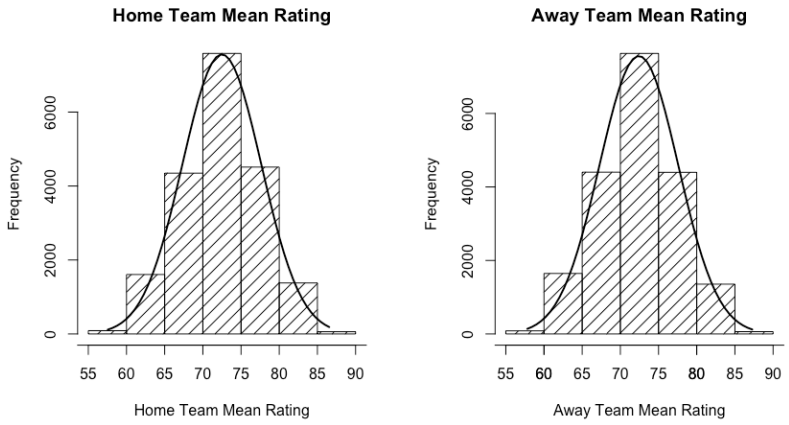


Figure 2.2: Histogram with the mean team ratings and the respective normal distribution

For the team's play speed we again have a rating once every year so we make the same assumption as with the player's rating. The distribution of the speeds can be seen in figure 2.3.

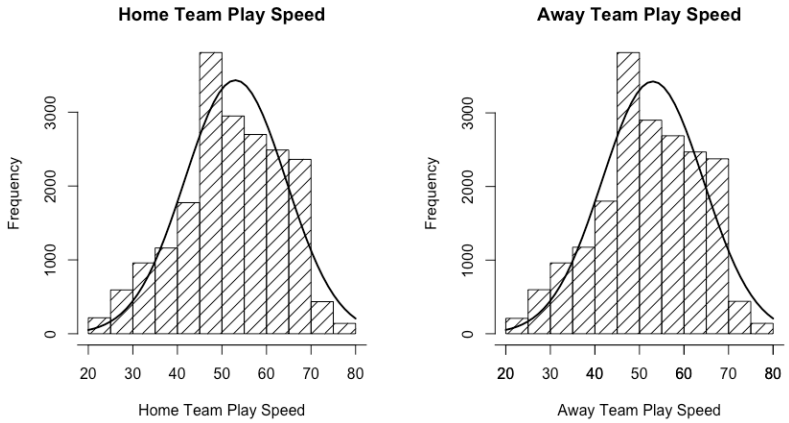


Figure 2.3: Histogram with the play speed for the teams per match

Besides statistics of both teams, which are constant during the year, it will be relevant to have more recent data. Consequently the recent results of both teams are useful. We take again our data set with the matches and sort it by date. Then for each team we look back to the last five matches to see what percentage of those matches they have won. We also want to know the recent history against the other team, so for this we look back to the last two matches that have been played against the other team. We take two because if we assume every team plays against each other home and away in one year, then the two most recent matches will be one away and one home game. Again we plot the histograms which are given in figure 2.4 and 2.5 respectively. What stands out is that we have a few matches where the

history of one of the teams is between 20% and 40%, 40% and 60%, 60% and 80% or 80% and 100%. These peculiar percentage gaps are there due to the fact that in the beginning the teams have not played five matches yet. So say a team has played three matches and won one then this will give the 33%. For the history against the same team there are only three possibilities: a team has won both of the matches against the other team, a team has won one of the matches or the team has lost or tied both of the matches against the other team.

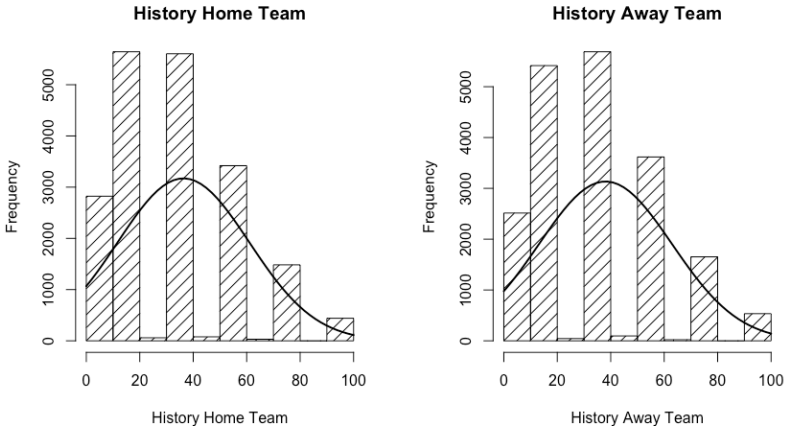


Figure 2.4: Histogram with the percentage of wins in the last 5 matches

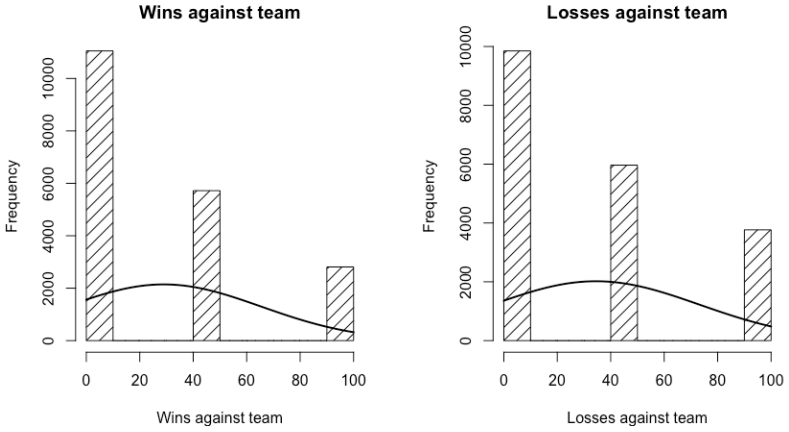


Figure 2.5: Histogram with the percentage of wins against the other team in the last two matches

2.1.1. Best betting company

To analyze which betting company predicts matches the best, we first need to convert the predicting odds to probabilities. Then we can take the highest percentage as a predicting result for the match. So for example for one of the matches and one of the bookkeepers we have the odds and probabilities as in table 2.2

Betting company	B365H	B365D	B365A
Betting odd	1.73	3.40	5.00
Probability	0.58	0.29	0.2
Percentage (%)	58	29	20

Table 2.2: Betting odds and probabilities for a random match

What is remarkable is that the probabilities do not add to one. This is because the betting

companies add an “over-round” [2] to their odd so that it will be less profitable for the person who wants to bet on the match. With these percentages we can compute the most likely outcome of the match. However, a curious result arose, all betting companies predicted around 53% of matches correctly. We can also clearly see this in the correlation plot, given in figure 2.6: on the one hand all betting companies are strongly correlated with each other, but on the other hand there is only a slight correlation with the real end result. This is illustrated by the small circular fractions. See appendix A for an explanation of the abbreviations.

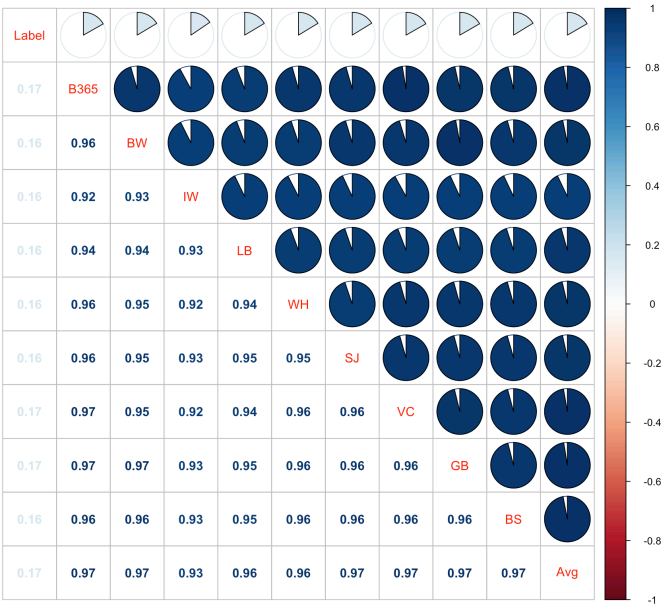


Figure 2.6: Correlation Plot for the betting companies

We will only use one of these betting companies to simplify the calculations. We choose to use BetVictor (VC)[3] because it has a slightly higher percentage than the other companies.

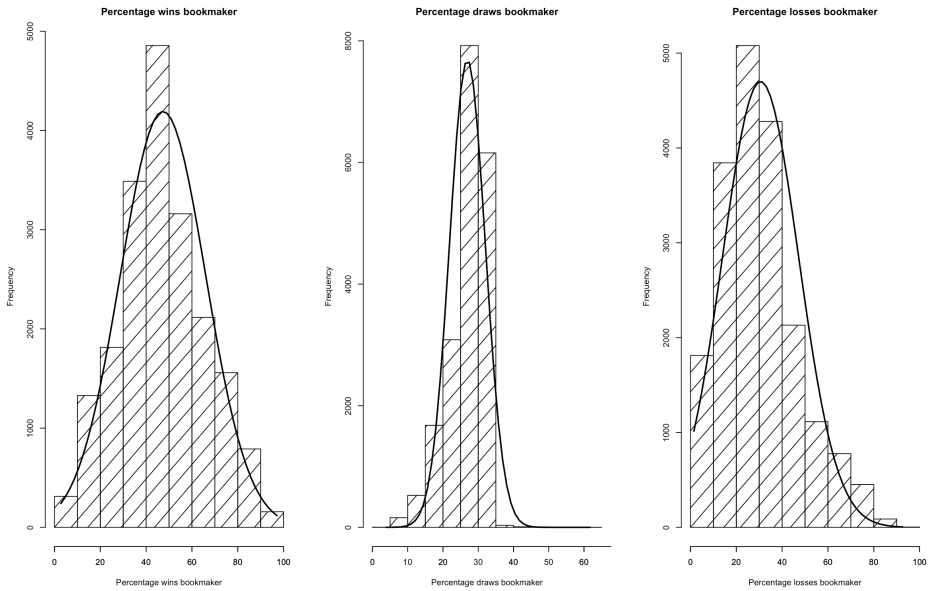


Figure 2.7: Histograms for the betting predictions

[2] http://www.bettingmarket.com/overround.htm
[3] https://www.betvictor.com/

2.2. Recap

With all these new variables we can create a new data set. When we first plotted a correlation plot, most variables were not correlated, so from that set we will only consider *away_team_goal*, *home_team_goal*, *label* and the betting company, which will be *VC*. Besides that we will consider the variables we just discussed and made a histogram of. Plotting the Pearson Correlation for these 14 variables results in figure 2.8. Immediately we see that the play speeds do not correlate with any of the other variables so we will not consider those in any further analysis. Besides that we can see one big red dot between *VC_Win* and *VC_Defeat* (actually two but the matrix is symmetric), this makes sense because when VC predicts a higher chance for a win, it has to have a lower chance for a defeat. The other circles that stand out are those between *home_goals* and *label*, which has a positive correlation and *away_goals* and *label*, which has a negative correlation. Again, this makes complete sense because if there are more home goals, it is likely that the home team won and if there are more away goals it is likely to be a defeat for the home team. Of course it can also be a tie with a lot of goals, or a win for one of the teams with just one goal difference, so this correlation is not very strong. The last thing that really stands out is that *VC_Win* and *VC_Defeat* are correlated with every other variable. This leads to the conclusion that this betting company takes into account what the current statistics for the teams are. We are now left with 19580 matches. Using the data set with all the countries, a table can be constructed which shows the matches per country, which is given in table 2.3

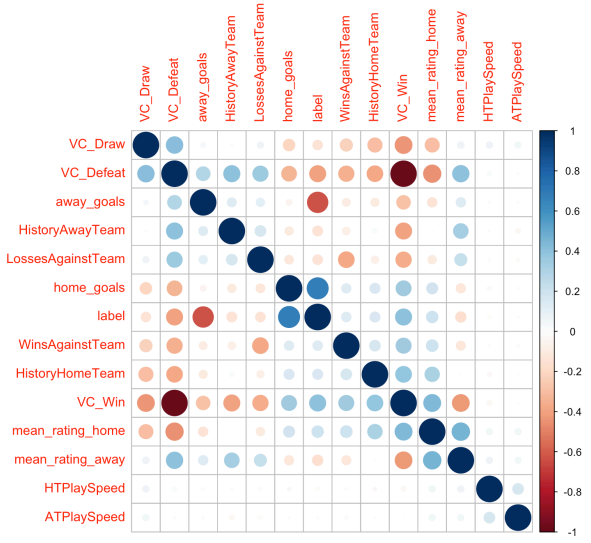


Figure 2.8: Correlation Plot

Country	Number of matches
Belgium	1196
England	2962
France	2861
Germany	2373
Italy	2723
Netherlands	2029
Portugal	1193
Scotland	1539
Spain	2704
Total	19 580

Table 2.3: Number of matches per country

Lastly we can also make histogram plots for the goals during the matches and the match result. These are given in figure 2.9 and 2.10. As we can see in the second plot, there are a lot more wins then there are losses or draws. Expressing this in percentages gives 46% wins, 29% losses and 25% draws. So if we would bet on a win we have a 46% chance of winning the bet. For the goals we see that there a lot of matches with either zero goals for the home team or zero goals for the away team and only a few matches with more than four goals for either team.

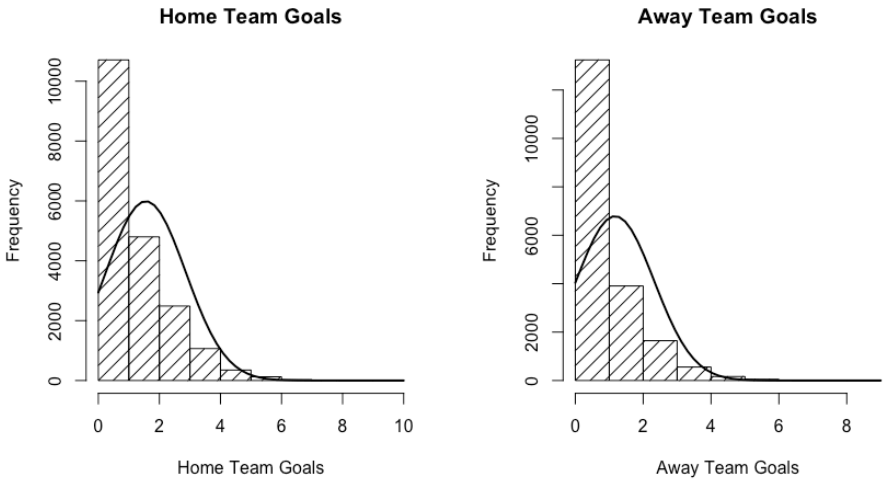


Figure 2.9: Goals distribution

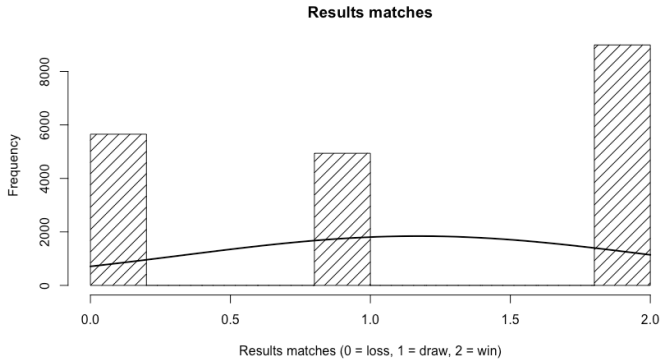


Figure 2.10: Result distribution

3

Bayesian Networks

In this chapter the theoretical background of Bayesian Networks is presented. The amount of studies done with Bayesian Networks has increased rapidly in the last decade, but it is all based on a couple of important definitions. In the first section the necessary probabilistic information is presented. In the second section we repeat some definitions coming from graph theory. Next we present the possible Bayesian Networks and in section 4 we see how we can learn these networks with the help of different algorithms. In the last section we briefly cover how the parameters of the network are fitted onto the graphical structure.

3.1. Probability theory

Before we can define a Bayesian Network, we will first need to introduce some concepts of probability theory. These definitions are adapted from Neapolitan (2004) [3]. The first one is independence and secondly independence for multiple variables, conditional independence:

Definition 3.1.1 *Two events A and B are **independent** if one of the following holds:*

1. $P(A|B) = P(A)$ *and* $P(A) \neq 0, P(B) \neq 0$
2. $P(A \cap B) = P(A)P(B)$

Definition 3.1.2 *Two events A and B are **conditionally independent** given C, denoted by $I(A, B|C)$, if $P(C) \neq 0$ and one of the following holds:*

1. $P(A|B \cap C) = P(A|C)$ *and* $P(A|C) \neq 0, P(B|C) \neq 0$
2. $P(A, B|C) = P(A|C)P(B|C)$

So for example if we roll two die, one blue and one red, then the two throws are independent of one another. But if you are then told that the throw with a blue die is not a six and the red one is not a one, then you can't gain any knowledge of the red die by looking at the blue one. In this case the throws are conditionally independent given the information. If, however, I tell you that the sum of both die is odd, and you see that the blue die is a 3, then with my information you have more knowledge of the throw of the red die. So now the two throws are not conditionally independent given the information. Next we also need the most important theorem for conditional probabilities, namely Bayes' Theorem:

Theorem 3.1.1 (Bayes) *Given two events A and B such that $P(A) \neq 0$ and $P(B) \neq 0$, we have*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.1}$$

which can also be extended with the law of total probability: when given events A_i such that $P(A_i) \neq 0 \ \forall i$ then for all $1 \leq i \leq n$:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B|A_1)P(A_1) + \ldots + P(B|A_n)P(A_n)} \tag{3.2}$$

Let's now turn to random variables. To compute the joint probability for random variables $X_1, \dots, X_n$ we know, by the chain rule (Proposition 2.1 [1]), that:

$$P(X_1, X_2, \dots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_2, X_1) \dots P(X_n|X_1, \dots, X_{n-1}) \tag{3.3}$$

However to compute this joint distribution involves computations for every conditional probability. This is one of the main strengths of Bayesian Networks, namely to represent this distribution in a much simpler way. But before we can represent our probability distribution we will first have to explore some definitions and theorems from graph theory.

3.2. Graph theory

A Bayesian Network is defined by a graph $G = (V, E)$, where V is the set of vertices and E the set of edges of the graph. We will only consider the edges to be directed, however it is also possible that these edges are undirected. If we look at our network we don't want any cycles to occur, so it should not be possible to start in one vertex and by following the arrows end up in the same vertex again. This means that our graph has to be a directed acyclic graph (DAG). Some other important definitions, again from Neapolitan (2004) [3]:

Definition 3.2.1 *Given a DAG $G = (V, E)$ and nodes X, Y in V*

- *Y is called a **parent** of X if there is an edge from Y to X*
- *Y is called a **descendent** of X and X is called an **ancestor** of Y if there is a path (a set of edges connecting two nodes) from X to Y*
- *Y is called a **non-descendent** of X if Y is not a descendent of X*

So, let's look at an example, taken from Koller and Friedman (2009) [2]. If we assume that a patient can have two diseases, Flu or Hay fever, then both these diseases have one plausible common cause, the season of the year. Both these diseases also have one common effect, a Congestion in the nostrils. Lastly we assume Flu causes Fever. The graphical representation of this model is given in 3.1. We can see that Season is a parent of Hay fever, but also that Season is an ancestor of Fever. Lastly we can also see that Congestion is a non-descendent of Fever.

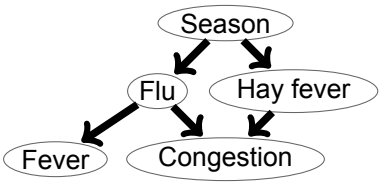


Figure 3.1: Easy example

As we can see in the graph, if we take three nodes, there are three possible combinations to connect these three nodes, which can be seen in figure 3.2.

(a) Serial connection (b) Convergent connection (c) Divergent connection

Figure 3.2: Different connections with 3 vertices

Now that we know what these connections look like we can define how connectiveness in a graph corresponds with conditional independence. However, a graph also encodes more conditional independence statements, which are generally characterized by the concept of d-separation:

Definition 3.2.2 *[5] If A, B and C are three disjoint subsets of nodes in a DAG G, then B is said to **d-separate** A from C if along every path between a node in A and a node in C there is a node v satisfying one of the following two conditions:*

- *v has converging arcs (i.e. there are two arcs pointing to v from the adjacent nodes in the path) and neither v nor any of its descendants (i.e. the nodes that can be reached from v) are in B*

- *v is in B and does not have converging arcs*

So again in the same example about sickness, we can see that Flu d-separates Fever from Congestion, but also that Flu d-separates Season from Fever. Which are exactly when we have a serial connection or a divergent connection! However Congestion does not d-separate Flu from Hay fever, so we don't have d-separation with a convergent connection. That's why this connection has a separate name, a **v-structure**, but only if there is no direct connection between Flu and Hay fever. Looking at the example again we can also determine that Congestion and Season are d-separated by {Flu, Hay fever}.

If we consider the graph where all directed edges are turned into undirected edges we are left with the **skeleton** of the graph. Two DAG's defined over the same set of variables are **equivalent** if they have the same skeleton and the same underlying v-structure.[4] An equivalent DAG for the example in figure 3.1 is given by figure 3.3b.

(a) Skeleton of the DAG (b) An equivalent DAG

Figure 3.3

With all these definitions we can define the most important property of Bayesian Networks: the Markov condition

Definition 3.2.3 *Suppose we have a joint probability distribution P of the random variables in some set V and a DAG $G = (V, E)$. We say that (G, P) satisfies the **Markov condition** if for each variable $X \in V$, $\{X\}$ is conditionally independent of the set of all its nondescendents given the set of all its parents. If we let PA_X the set with the parents of X and ND_X the set with the nondescendents of X then we can write $I(\{X\}, PA_X | ND_X)$.*
If we use the Markov condition for our joint distribution it will look like

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(X_i | PA_{X_i}) \tag{3.4}$$

which saves a lot of time to compute the conditional probabilities.

If we take for example the graph in figure 3.1, we can see that *Fever* is conditionally independent of *Congestion* given *Flu*. The joint probability distribution for this graph will thus be

$$P(S, H, C, Fl, Fe) = P(S)P(Fl|S)P(H|Fl, S)P(C|H, Fl, S)P(Fe|C, H, Fl, S) \tag{3.5}$$
$$= P(S)P(Fl|S)P(H|S)P(C|Fl, H)P(Fe|Fl) \tag{3.6}$$

We have now seen that we can factorize our joint distribution in a much simpler way, which involves a lot less computation for the conditional probabilities. This fact is also exploited by the algorithms that we will see in a later section, which we will then use to learn the structure of the Bayesian Networks. Before we can do this, we will look at what type of Bayesian Networks are available.

3.3. Types of Bayesian Networks

There are three types of Bayesian Networks. The simplest form is the discrete Network, where all nodes have a discrete probability distribution, which is given by a conditional probability table. Most discrete networks are modelled with a binary variable, so something is either yes or no. If we let all our nodes in the previous example be binary, we can write a conditional probability table for Fever as in table 3.1.

		Flu	
		Yes	No
Fever	Yes	0.6	0.2
	No	0.4	0.8

Table 3.1: Conditional probability table for Fever

Besides discrete networks there are also continuous Bayesian networks. To use a continuous network there is one important assumption to be made, namely that every variable is normally distributed and that the joint probability distribution is multivariate normal. We make this assumption because it makes the computations a lot easier and although most of the time the data does not satisfy this assumption, the generated network can still give a good approximation. The normal distribution for one variable is given by:

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.7}$$

where μ is the mean of the distribution and σ^2 the variance. As we saw in figures 2.2 till 2.10 not all of our variables are normally distributed.

Lastly there are also hybrid networks. These exists of discrete and continuous nodes. The problem is that these hybrid networks are very complex and the algorithms that exist to solve these kind of problems have a lot of limitations.
Bayesian Networks are graph structures with for every node a probability distribution. However, as we can see in table 3.2 there are a lot of possible DAG's for the 12 variables that we have. These cannot be constructed by hand, so we will have to look at different learning algorithms.

Nodes	Number of DAGs
1	1
2	3
3	25
4	543
5	29281
⋮	⋮
10	$4.2 \cdot 10^{18}$
11	$3.2 \cdot 10^{22}$
12	$5.2 \cdot 10^{26}$

Table 3.2: Number of possible graphs generated for given amount of nodes [1]

3.4. Structure Learning

To create a Bayesian Network from a data set we will have to make use of algorithms. All these algorithms make use of two steps: structure learning and parameter estimation. For structure learning the algorithms can be divided into three categories: constraint- and score based learning and and a combination of the two, the hybrid algorithms. We can also combine these algorithms with the knowledge of experts. These experts know some relations between the variables.

3.4.1. Constraint based learning algorithms

Constraint based algorithms learn the network by making use of conditional independence statements. With these statements the algorithm can learn the skeleton of the network and the next step is to identify the v-structures in the network. The main structure of these algorithms is given by:[1]

Algorithm 1 The structure of a Bayesian network can be learned from conditional independence statements of the form, "A independent of B given C" (denoted by $I(A, B, C)$):

function LearnStructure(S)
Find the skeleton of the Bayesian network:
The link $A - B$ is part of the skeleton if and only if $\neg I(A, B, X)$ for all X not containing A or B.
Direct the links:
if A, B, C such that $A - C$ and $B - C$, but not $A - B$ **then** introduce the v-structure $A \rightarrow C \leftarrow B$ if there exists an X (possibly empty) such that $I(A, B, X)$ and $C \in X$.
else if $A \rightarrow C - B$ (and no link between A and B) **then** direct $C \rightarrow B$
else if $A \rightarrow B$ introduces a directed cycle in the graph **then** do $A \rightarrow B$
else choose an undirected link and give it an arbitrary direction
end if
end function

To test the conditional independence statement, we make use of statistical independence tests such as the t-test or the Mutual Information. The t-test $t(X, Y|Z)$ is given by

$$t(X, Y|\mathrm{Z}) = \rho_{XY|\mathrm{Z}} \sqrt{\frac{n - |\mathrm{Z}| - 2}{1 - \rho^2_{XY|\mathrm{Z}}}} \tag{3.8}$$

where X and Y are random variables, |Z| a set of random variables and $\rho_{XY|\mathrm{Z}}$ the partial correlation coefficients. The Conditional Mutual Information is given by:

$$I(X, Y|Z) = \sum_{z \in Z} \sum_{y \in Y} \sum_{x \in X} p_{X,Y,Z}(x, y, z) \log \left(\frac{p_Z(z) p_{X,Y,Z}(x, y, z)}{p_{X,Z}(x, z)\, p_{Y,Z}(y, z)} \right) \tag{3.9}$$

where $p_{X,Y,Z}(x, y, z)$ is the joint probability mass function of X, Y and Z and $p_X(x)$ and $p_Y(y)$ are the respective marginal probability mass functions.
Some well used constraint based algorithms are Grow-Shrink (gs) and Fast- and Inter Incremental Association (Fast-IAMB and Inter-IAMB). We will also use these three in our further analysis. As we saw before, for twelve variable there are $5.2 \cdot 10^{26}$ possible structures. To reduce this amount, the algorithms first look at what the **Markov blanket** of a node is. The Markov blanket consists of the parents, children and parents of the children of the node. The consequence of this step is that we have to do a lot less conditional independence tests, which in turn leads to a decrease in computation time.

3.4.2. Score based learning algorithms

Score based algorithms work quite simple, they give each network a certain score and the one with the highest score is returned. This means that a couple of networks are selected by the algorithms and each of them is assigned a score, which is actually a goodness of fit statistic measuring how well the DAG corresponds to the dependence within the data. The algorithms that we will use, Hill-Climbing (hc) and Tabu, will start with an empty graph, so just the nodes, and add, delete or reverse an arc until the score is optimized. The score that we will use is BIC (Bayesian Information Criterion) given by [1]:

$$BIC(S|\mathcal{D}) = log_2 P(\mathcal{D}|\hat{\theta}_S, S) - \frac{size(S)}{2} log_2(N) \tag{3.10}$$

where $size(S)$ is the number of independent parameters in S, $\mathcal{D}$ the database, $\hat{\theta}$ an estimate of the maximum likelihood parameter and N the number of data points per variable. The number of independent parameter for one node is equal to two plus the number of parents of the node. So for example $size(S)$ for the network in figure 3.1 is equal to 13.

3.4.3. Hybrid learning algorithms

Hybrid learning algorithms combine the strengths of both the previous algorithms. Consequently, these algorithms consist of two steps: first the amount of possible structures is restricted with the help of independence test and the second step is to maximize the score function restricted on this set of structures. We will use the Max-Min Hill-Climbing (MMHC) algorithm, which is a combination of the constraint based Max-Min Parents and Children and the score based Hill-Climbing algorithm.

3.4.4. Experts

Besides using algorithms to construct our network, we can also consult experts in the field. So a doctor for example will know what kind of symptoms belong to a certain disease and what causes this disease. We can also combine the expert knowledge with our algorithms to discover new relationships that the doctor might not know about, but which arise from a data set with patients. This might be useful to discover new diseases or other symptoms belonging to the same disease that the doctor does not know about. These experts can also point out variables which are not considered in the model but which could influence the predictions.

3.5. Parameter fitting

Now that we know what our structure looks like, we can fit the parameters onto the different variables. This fitting is greatly simplified because we can decompose the global distribution into the local distributions by making use of the **Markov Condition**, as seen in definition 3.2.3. The parameter estimation is done by making use of maximum likelihood estimation (mle). This method uses the likelihood of a Bayesian Network $M(S,\theta)$, where S is a structure and θ the parameters, given the data D, computed by[1]:

$$L(M|D) = \prod_{d \in D} P(d|M(S,\theta)) \tag{3.11}$$

where $P(d|M)$ is the likelihood of M given d. The goal is to maximize this function, so find θ such that $L(M|D)$ is maximal. If we have a discrete data set this means that we will have a conditional probability table for each of the nodes and if we use a continuous approximation of the data we will have the regression coefficient for each variable against its parents.

4

The Bayesian Network Football Structure

Now that we have seen what our data looks like and what Bayesian Networks are, we can combine these two together. We have seen that there are multiple algorithms to determine the structure of our Bayesian Network. The most intuitive thing to do, is just apply some of these algorithms on the whole data set and see what these structures look like. We will do this first for the constraint based algorithms, which results in figure 4.1 and 4.2. The red lines indicate that the direction of the arc is different then the one in the base model, which here is the Grow-Shrink one, while the blue lines represent lines that are present in the first model, but not in this model. We can clearly see that the networks learned by the constraint based algorithms look quite similar. They have a lot of black lines which means that the arc is present and has the same direction. Secondly, we can also observe that the network learned by the Grow-Shrink algorithm is exactly identical to the one learned by fast IAMB, but also that that the IAMB and the inter IAMB network are identical.

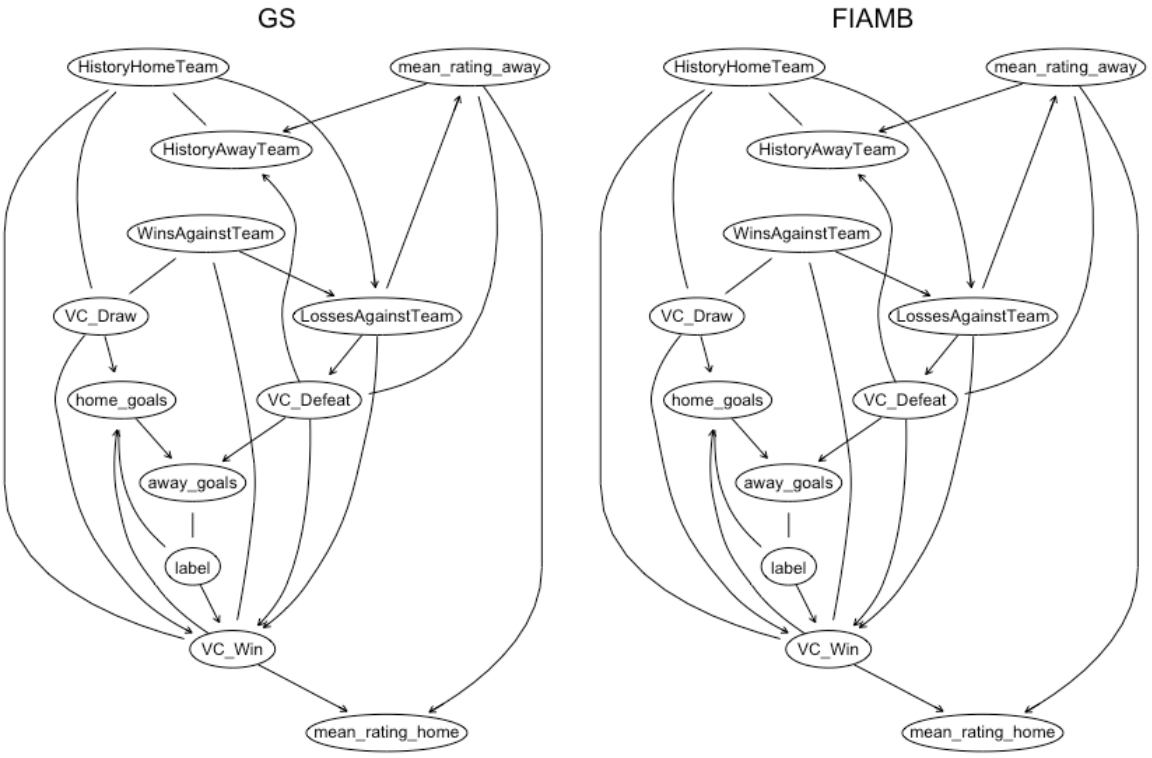


Figure 4.1: Networks learned by the Grow-Shrink and Fast IAMB algorithms for the full data set

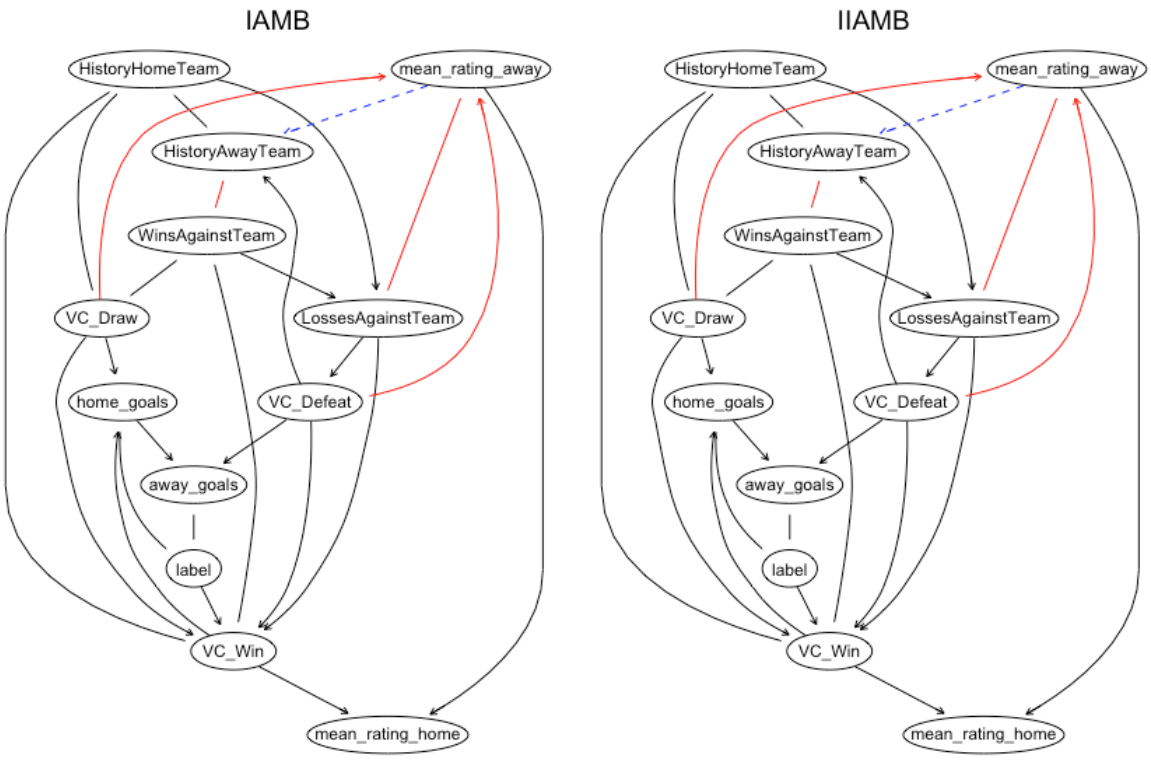


Figure 4.2: Networks learned by the IAMB and Inter IAMB algorithms for the full data set

If we compare the score and hybrid methods, however, we can see that they are almost completely different from the structures learned by the constraint based algorithms, as can be seen in figure 4.3 and 4.4. On the other hand, both score based structures are almost identical, with only two different arcs, as indicated by the red and blue lines in the tabu structure.

With just these seven networks there are already a few things that stand out. For one, we see that the betting nodes *VC_Win*, *VC_Draw* and *VC_Defeat* take up a central position in all of the networks. This could mean that they play an important role in predicting our label or the home and away goals. Secondly the structures learned by the score based algorithms contain a lot more arcs than the ones learned by the constraint based algorithms. This can be explained by the fact that the constrained based algorithms first look at the skeleton of the graph, which is determined by the conditional independence of two nodes given another one. The score based algorithms on the other hand, add an arc each time to optimize their score, so it will just continue adding arcs if it increases it's score. Lastly, we can also see that there are undirected edges in the structures learned by the constraint based algorithms. This is something we would like to avoid, because otherwise it would not be possible to learn the parameters of the variables connected by these edges.

This gives us an idea of what the structures will look like, but we want to know what the influence will be if we change our data a little bit. This can be tested by making use of a bootstrap method.

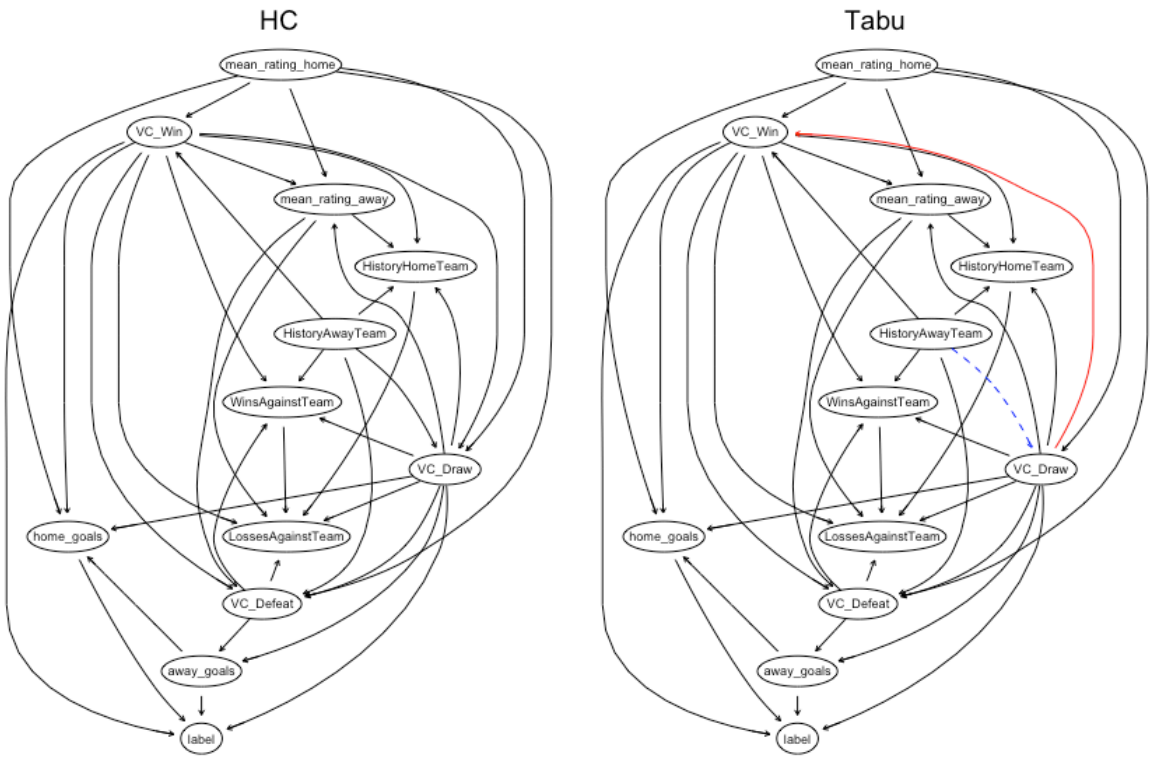


Figure 4.3: Comparison between the two score based algorithms

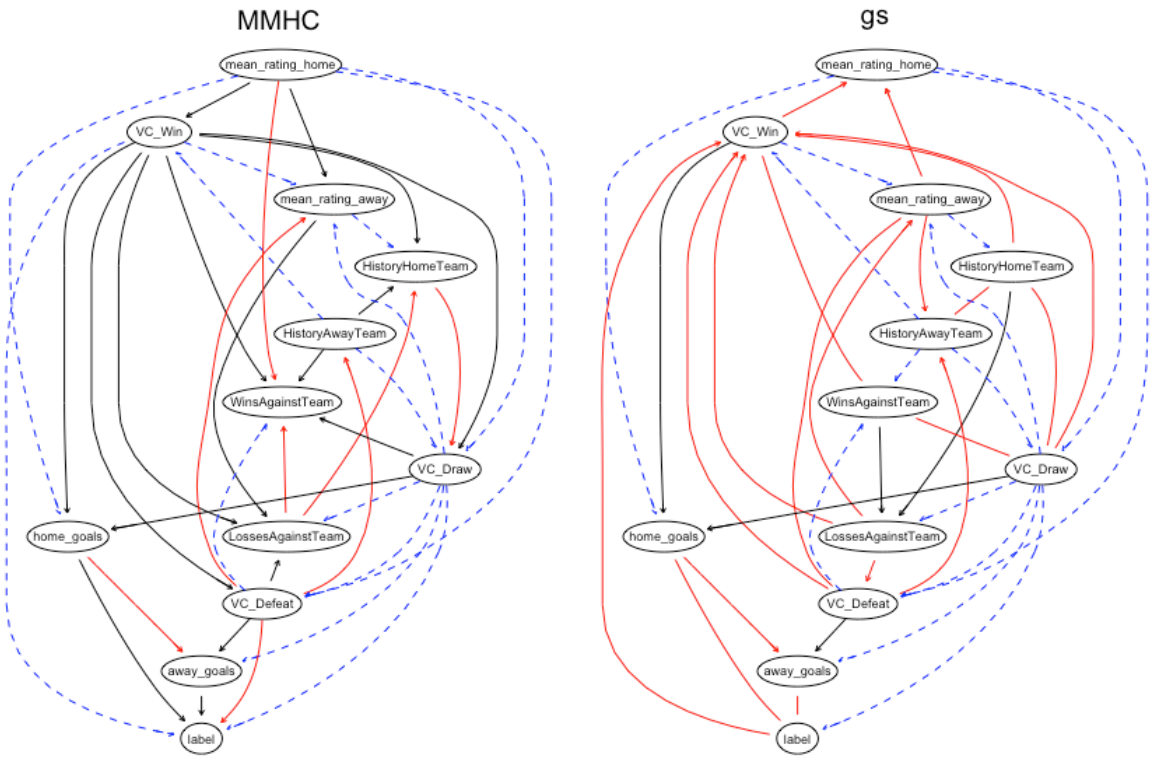
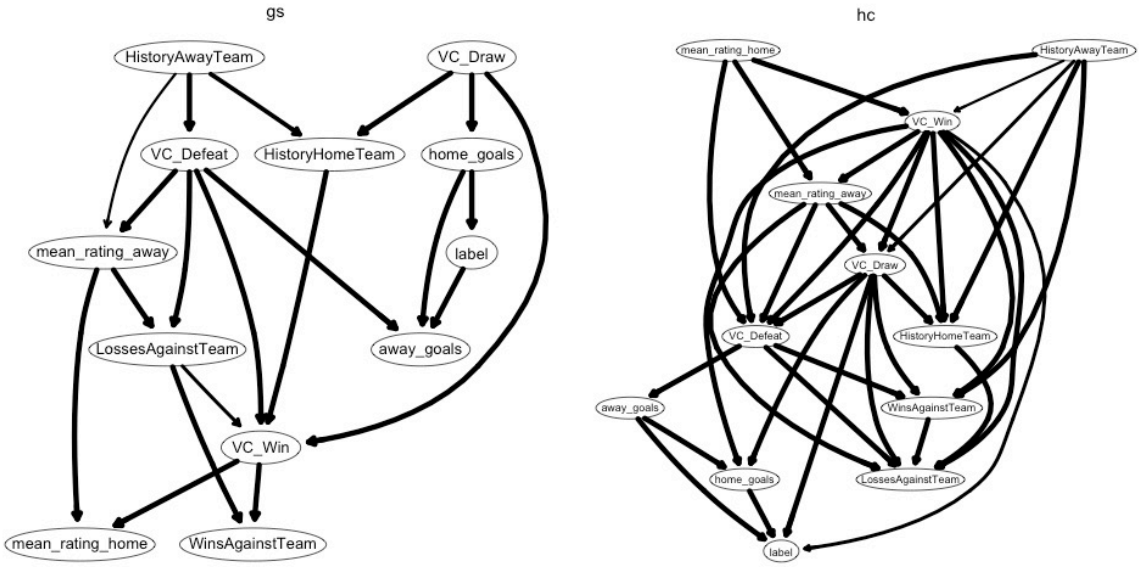


Figure 4.4: Comparison between MMHC and GrowShrink with relation to HillClimbing

4.1. Bootstrap

The bootstrap method is a statistical test based on random sampling with replacements. What this means, is that if we have a vase with 5 coloured balls, then we randomly pick 5 of those balls but each time we put the ball back. So say, for example, that we have a yellow (y), a red (r), a blue (b), a green (g) and a white (w) ball. Then bootstrapping this data could result in the following set: y-w-y-r-w. For our purposes, the structure is then learned on this set. This was only one step, but we can repeat this process to get multiple structures and we can check which of the arcs in the structure occurs most of the time. Repeating this process 50 times for our data set, and only taking the arcs which occur more than 70% of the time, results in figure 4.5a for the Grow-Shrink algorithm and figure 4.5b for the Hill-Climbing algorithm. The first thing that stands out is that the Grow-Shrink network has again far less arcs than the Hill-Climbing network, which again can be explained by the fact that the links that are present in the Hill-Climbing network and not in the Grow-Shrink structure, have a low conditional independence as tested by equation 3.8. Secondly, we also see that some arcs are less thick than others. This is due to the fact that they occur less often with respect to the others. So for example the link from *HistoryAwayTeam* to *mean_rating_away* occurs only 80% of the time, while the other arcs occur almost 100% of the time.



(a) Strength plot for the Grow-Shrink (b) Strength Plot for the Hill-Climbing

Figure 4.5: Strength plots for Grow-Shrink and Hill-Climbing, based on a bootstrap method

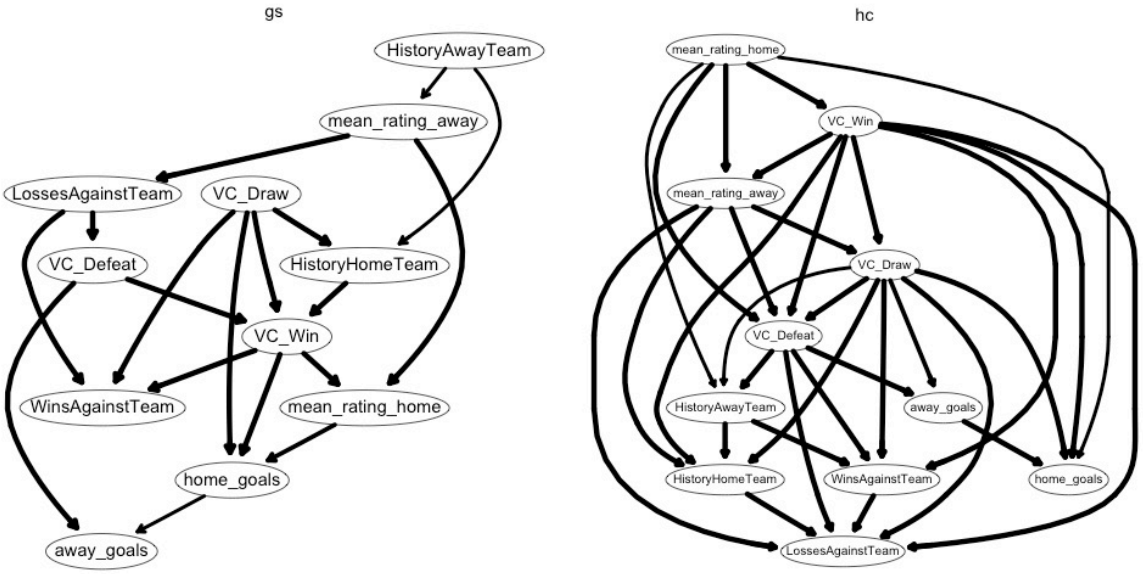
However, our goal was to do predictions but by using all of our data to learn a structure, we do not have any data left to test our structure on and to see how well our predictions were.

4.1.1. Predicting

To validate if the model that we use fits the data set, we first have to divide the data into two parts, one training set (usually 80%) and one test set (usually 20%). For the training set we have to sample (so pick randomly) 80% of the data and the test set is the other 20%. We also want to remove the variable *label* from both these data sets because if we have our home and away goals we can use those to deduce the label. With these modified data sets we can again apply the bootstrap method as in the previous section, repeating the process of sampling with replacement 50 times and only taking the arcs which occur more than 70% of the time.

The first thing we notice is that the graphs generated by the constraint based algorithms and the score based algorithms are again completely different. For Grow-Shrink, Fast IAMB

and Inter IAMB there are a lot of arcs pointing towards *VC_Win*, which in turn has an arc towards *home_goals*, while Hill-Climbing and Tabu just have a lot of arcs pointing in every direction. The graph generated by MMHC is just a bit in the middle, which we expect because it is a combination of the two. The plots for both Grow-Shrink and Hill-Climbing are below and the others can be found in appendix B.1.



(a) Strength Plot for the Grow-Shrink (b) Strength Plot for the Hill Climbing

Figure 4.6: Strength plots for Grow-Shrink and Hill-Climbing based on 80% of the data and a bootstrap method

Now that we know what our graph looks like, we can also fit the parameters belonging to each variable. So for example for the variable *VC_Defeat* in the Grow-Shrink structure we get the following table:

Parameters of node VC_Defeat (Gaussian distribution)	
VC_Defeat | LossesAgainstTeam	
Coefficients:	
(Intercept)	LossesAgainstTeam
25.098513	0.161494
Standard deviation of the residuals: 15.38499	

Table 4.1: Parameter for node mean rating away in the Grow-Shrink model

But what does this actually mean? It says that *VC_Defeat* is normally distributed with mean $25.10 + 0.16 \cdot$ LossesAgainstTeam and standard deviation 15.38, or compactly written: $VCD|LAT = lat \sim N(25.10 + 0.16lat, 15.38^2)$. So we can make a linear model for *VC_Defeat* which depends only on *LossesAgainstTeam*.

Next we can use the learned structure and it's corresponding parameters to do some predictions. This is done by using the function *predict* in the package bnlearn[1]. The function averages likelihood weighting simulations by using the evidence given in the test data set. For each match in the test set this is done by computing the conditional distribution given all the other nodes and calculating the expected value.

Because of our assumption that the variable follows a normal, so continuous distribution, the resulting predicted values will be real numbers, so most likely not integers. However, it is of course not possible to have 1.4563563 goals, so we will have to round the numbers. Lastly we want to compute how well our predictions are. Therefore, we calculate the Mean Square

[1] bnlearn.com

Error between the prediction and the real variables. The Mean Square Error is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1}$$

where n is the number of predictions, so 20% of 19580, Y_i are the real values and $\hat{Y}_i$ are the predicted values. The errors are given in the following table:

	HC	GS	FIAMB	IIAMB	Tabu	MMHC
MSE HomeGoals	1.471	1.473	1.480	1.477	1.474	1.466
MSE AwayGoals	1.153	1.152	1.152	1.153	1.155	1.153

Table 4.2: Mean Square Error for the different algorithms

As we can see the errors are all relatively small and close to one and other, but the error for the away goals is smaller than for the home goals. This can be explained by the fact that the away goals have a smaller variance than the home goals and will thus be in a smaller region around the mean. The relatively small error for both variables is due to the fact that we have a test set of around 4000 matches, while there is only a small difference between the observed and predicted goals. With the predicted home and away goals we can also determine what the end result of the match was and thus how well it compared to the real end result. We compute this by just comparing which team scored the most, so not the exact outcome of the match.

	HC	GS	FIAMB	IIAMB	Tabu	MMHC
% correct	52.58	51.81	51.53	52.37	52.27	51.81

Table 4.3: Percentage of matches predicted correctly for the different algorithms

Again we see that all algorithms are quite close to one another with their predictions. And also that all percentages are a bit lower than the percentage of our betting company, which was 53%.

By now we haven't consulted any experts, so all arcs are learned by the algorithm. However we could for example state that there should not be an arc between *home_goals* and *away_goals*. Lastly, we do have to mention that running our bootstrap method on different 80% samples of the data gave different significantly different structures. We can compensate for this and validate our model by using K-fold cross validation.

4.2. K-fold Cross Validation

For our bootstrap method we randomly sampled 80% of the data, but why 80% and not 60 or 75? And although we sample randomly, we have a fixed test set to perform our predictions on. If we decrease the size of our training set, the learned model will be less accurate. To combat this ambiguity we make use of K-fold cross validation. K can be any integer within the amount of matches available, but we choose to use 5 to have a connection with the 80% as picked in the bootstrap method. The idea is that the data set is split up into 5 folds. The network is then learned on 4 of the 5 folds and tested on the remaining fold, as visualized in figure 4.7.

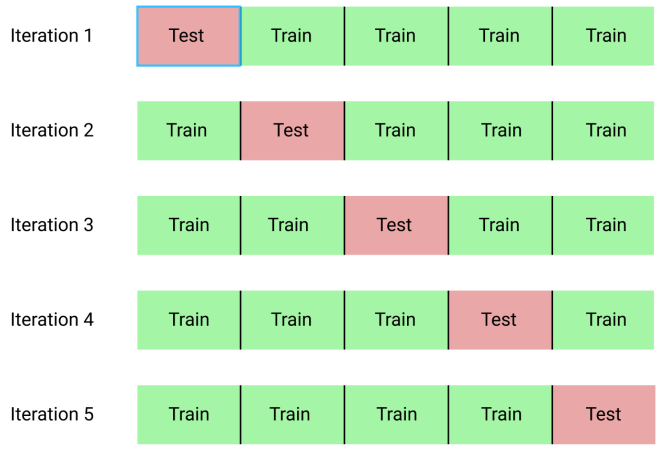


Figure 4.7: Visualization of 5-fold cross validation [6]

Due to the structure of this method, it is only possible to predict one variable, so we will remove both the variables *home_goals* and *away_goals*, and only predict the end result of the match, so the label. The error is again computed by the mean square error, as given in equation 4.1. This method can again be repeated multiple times to get a more accurate result.

This is the way it should work for most data sets, however the structure learned by the constraint based algorithms results in an error because there are undirected arcs around the node we want to predict, as we also saw in our first network. We will come back to this when we use our expert knowledge. However, for score and hybrid based algorithms it does work. So we repeat this 5-fold cross validation 50 times and per time compute the percentage that was predicted correctly. We can put the prediction percentages in a table and the mean square errors in a box plot. These are in table 4.4 and figure 4.8 respectively. Besides the minimum, maximum, mean and median, the table also states the first and third quantile. The first quantile splits off the lowest 25% of the data, while the third quantile splits off the highest 25% of the data. As we can see, the results are not that promising. The networks only predict around 37% of the matches correctly. However, we can also see that the mean square error is quite small for the three different algorithms and even smaller than the mean square error of the bootstrap method. This is due to the fact that the difference for the label can be maximally 2, while the difference for the goals could theoretically be 10, which give an enormous increase when squaring it. It can also be observed that the predictions made with our bootstrap method are a lot higher than those with the cross validation. This is probably due to the fact that we have more information, namely both goals, instead of just one number for the end result. To improve we will have to make use of an expert who tells us which arcs will be in the network.

	Min	1st Quantile	Median	Mean	3rd Quantile	Max.
Tabu	37.46	37.64	37.68	37.68	37.73	37.88
HC	37.37	37.67	37.72	37.72	37.77	37.94
MMHC	36.72	37.04	37.15	37.14	37.28	37.49

Table 4.4: Percentage of matches predicted correctly for score and hybrid based algorithms

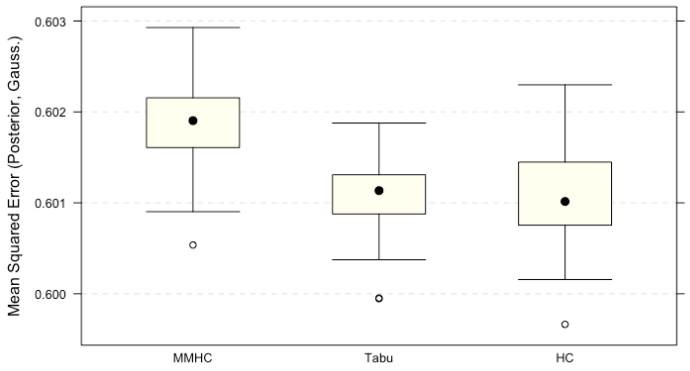


Figure 4.8: Mean square errors for 50 runs with score and hybrid based algorithms

4.3. Combining bootstrap graph with cross validation

We can also combine our bootstrap method with the cross validation by instead of learning the structure on a part of the data, using the bootstrap method to learn a structure and then learn the parameters on each fold. Implementing this method by doing a bootstrap 50 times and 10 5-fold validations, instead of 50 for the sake of computing time, gives the predictions for the label in table 4.5 and the mean square errors in figure 4.9. As we can clearly see the box plot looks more like a line plot due to the fact that the error difference is relatively big between the constraint and score based algorithms. Similarly the predictions by the score based algorithms are a lot better than those by the constraint based algorithms. The box plots for the score based algorithms are comparable to those in figure 4.8. Unfortunately, the predictions are worse than those with a standard cross validation. Lastly we also note that the prediction for the fast and inter IAMB methods are exactly the same. This is probably due to the fact that these algorithms are based on the same algorithm and will thus give similar structures and corresponding parameters.

	Min	1st Quantile	Median	Mean	3rd Quantile	Max.
Tabu	36.70	36.76	36.78	36.80	36.83	36.92
HC	36.53	36.71	36.74	36.75	36.81	36.91
MMHC	36.58	36.66	36.73	36.73	36.79	36.89
GS	30.88	30.98	31.08	31.03	31.10	31.12
FIAMB	28.87	28.87	28.87	28.87	28.87	28.87
IIAMB	28.87	28.87	28.87	28.87	28.87	28.87

Table 4.5: Percentage of matches predicted correctly for score and hybrid based algorithms

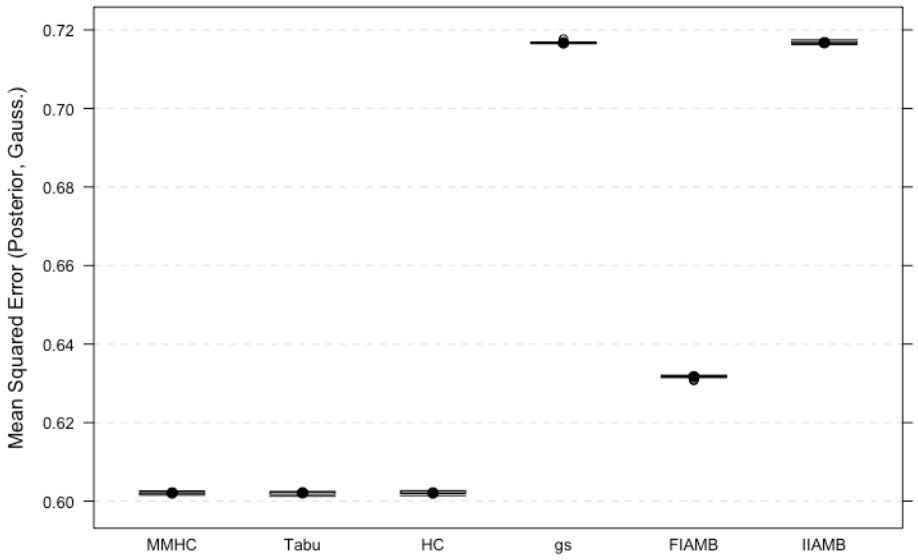


Figure 4.9: Mean square errors for 10 runs

4.4. Expert knowledge

We have seen in the bootstrap method that the goals are mostly influenced by the betting companies. But what if we let all arcs point towards the home and away goals and no arc between those? This will let the goals be dependent on all other variables, and while it will only be a small influence it could increase the predictions. Secondly, we can see what the influence will be if we remove the betting predictions from our data. The resulting structure then only contains the history of both teams and their current rating.

4.4.1. Bootstrap

The first thing we will do to improve our predictions is to let every variable influence the goals and then letting the fitting decide with what weight each variable influences the goals. Using the same procedure as before: doing 50 runs and taking an edge if it occurs more than 70% of the time. The results are really promising. If we look at table 4.6 we can see that the amount of matches predicted correctly is higher than the ones we predicted without any fixed arrows and also higher than the predictions made by the betting company. What also stands out is that the prediction percentages for all algorithms are close to one another, they are all within the range $(54.3, 54.8)$. This implies that we have found a structure which gives consistent predictions for all algorithms. However, we have forced a lot of arrows from the start so the algorithms don't make a big difference in the structures. When plotting the learned structures for the Grow-Shrink and the Hill-Climbing model in figure 4.10 and 4.11, we see again that the Hill-Climbing network contains a lot more arcs. The red arrows in the structures indicate the arrows that we forced to exist. In appendix B.2 the graphs for the other algorithms can be found.

	HC	GS	FIAMB	IIAMB	Tabu	MMHC
MSE HomeGoals	1.476	1.478	1.475	1.474	1.476	1.477
MSE AwayGoals	1.153	1.153	1.155	1.154	1.154	1.153
% correct	54.72	54.60	54.52	54.49	54.34	54.60

Table 4.6: Prediction results with all variables directed to the goals

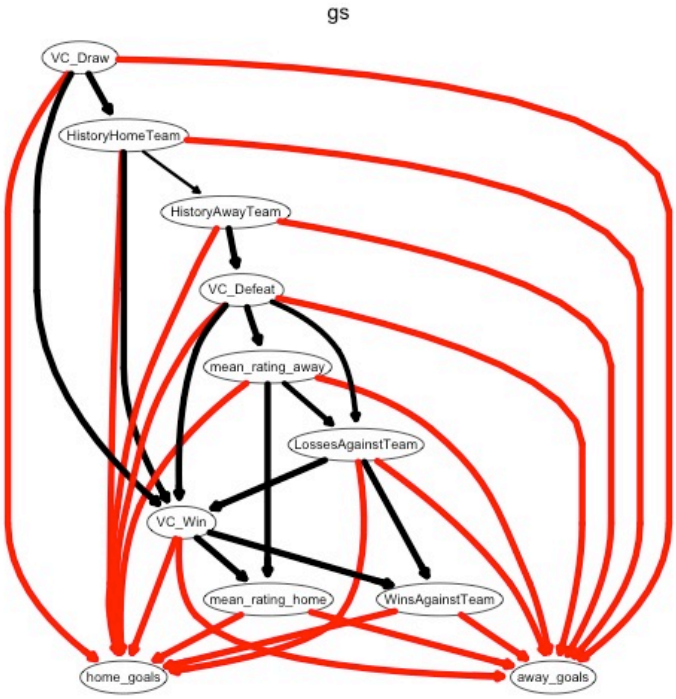


Figure 4.10: Strength Plot for GrowShrink with fixed arcs

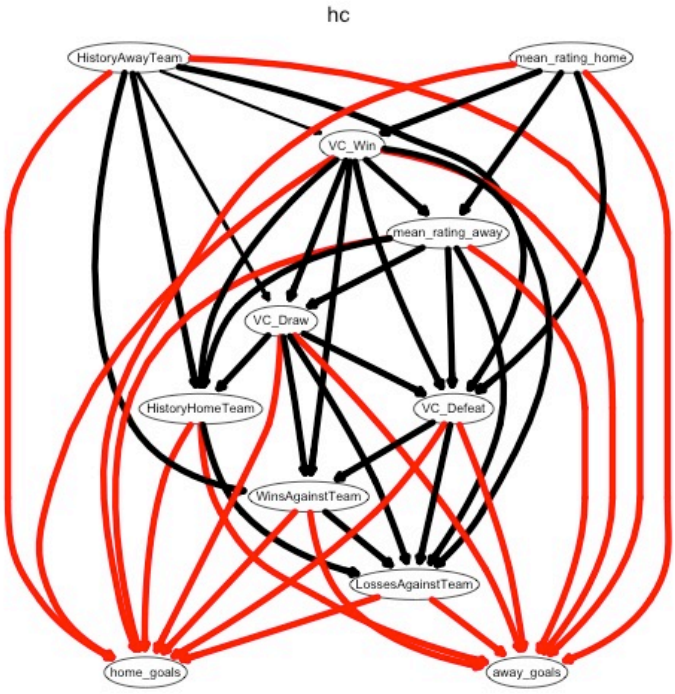


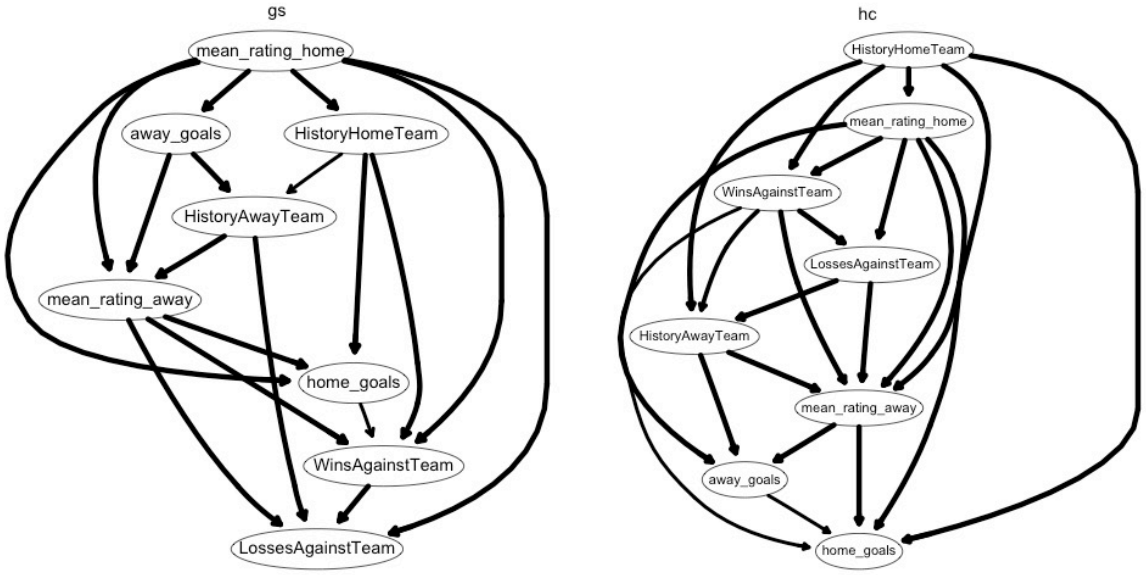
Figure 4.11: Strength Plot for Hill Climbing with fixed arcs

Next we will remove the betting nodes. We saw that they are always quite in the centre of the network so they will probably have a lot of influence. This is also the case when we do our predictions, they are lower than both the original network and the previous network. We can also see in figure 4.12a that the goals are not the end nodes of the network, which means that they also influence the history which is quite strange and of course does not occur in

the real world as the history of the team is computed before the match is played. In appendix B.3 the networks for the other algorithms can be found.

	HC	GS	FIAMB	IIAMB	Tabu	MMHC
MSE HomeGoals	1.489	1.494	1.475	1.499	1.491	1.496
MSE AwayGoals	1.189	1.204	1.155	1.210	1.189	1.189
% correct	49.77	50.41	50.46	50.87	49.51	51.2

Table 4.7: Prediction results by removing the betting nodes from the structure



(a) Strength Plot for Grow-Shrink

(b) Strength Plot for Hill Climbing

Figure 4.12: Strength plots for GS and HC without the betting nodes

4.4.2. K-fold cross validation

For the cross validation we can apply the same restrictions to the arcs as we did with the bootstrap method. So firstly we direct all the arc towards label, unfortunately the constraint based algorithms still give the error of undirected arcs so we will not be able to use those. Hence we do again a 5-fold cross validation and repeat this method 50 times to get the best result. The correctly predicted percentages are in table 4.8 and the box plot for the mean square error is in figure 4.13. A few remarks can be made. Firstly, the median error, which is the black dot in the box, of the 50 runs is lower for all algorithms than it was without the forced arcs, though the difference is really small. On the contrary, the percentage of the predictions predicted correctly is a little bit lower than it was before, but again this difference is really small. This implies that forcing the arcs towards the label has a negligible effect on the predictions.

	Min	1st Quantile	Median	Mean	3rd Quantile	Max.
Tabu	37.55	37.69	37.74	37.74	37.79	37.93
HC	37.58	37.68	37.78	37.78	37.87	37.98
MMHC	37.59	37.72	37.79	37.79	37.85	37.97

Table 4.8: Percentage of matches predicted correctly for score and hybrid based algorithms and all variables having an arc directed towards label

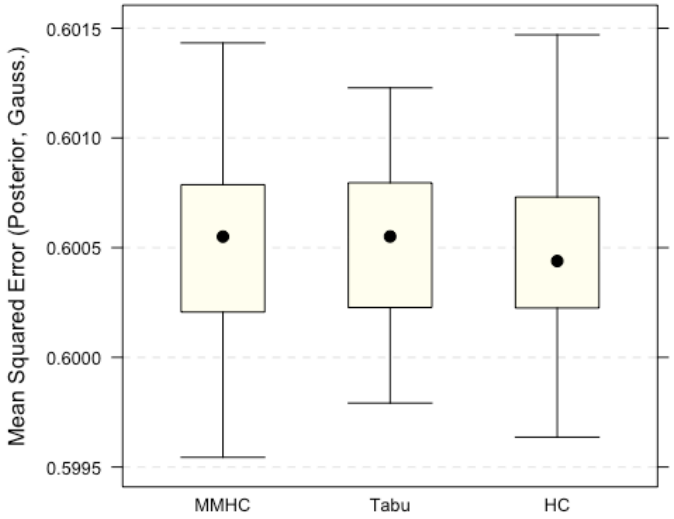


Figure 4.13: Mean square errors for 50 runs with score and hybrid based algorithms and all variables having an arc directed towards label

Next we remove again the betting nodes from the data set. Immediately we see a difference occurring, because the mean square error is bigger than it was in previous variations, as can be see in figure 4.14. Now we have a mean square error of 0.6167, while the previous variations were around 0.601. Again, this difference doesn't look that big but we have to remember that we have a big data set and a small difference between the real and observed value. As with the bootstrap method, by removing the betting nodes we have slightly decreased our predictions performance by now only predicting 35% correctly which previously was 37%. This was also something we observed by looking at the structures and the position of the betting nodes.

	Min	1st Quantile	Median	Mean	3rd Quantile	Max.
Tabu	34.87	35.08	35.11	35.12	35.17	35.28
HC	34.91	35.05	35.11	35.11	35.17	35.30
MMHC	34.96	35.08	35.18	35.16	35.22	35.34

Table 4.9: Percentage of matches predicted correctly for score and hybrid based algorithms without the betting nodes

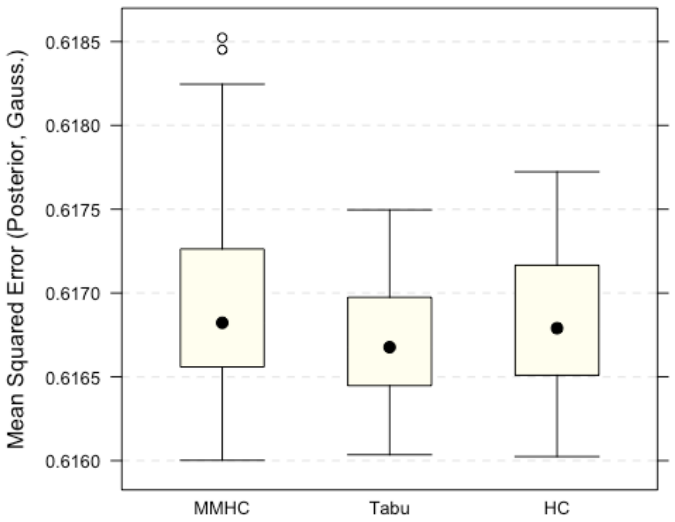


Figure 4.14: Mean square errors for 50 runs for the score and hybrid based algorithms without the betting nodes

4.4.3. Combining bootstrap graph with cross validation

As we saw in both methods, if we direct all arcs to the nodes we want to know we get the best results. So combining again the bootstrap structure and the cross validation gives the results in table 4.10 and figure 4.15. A peculiar thing stands out, the errors and predictions are very close to one another. There is almost no difference between the score and constraint based algorithms, though on average the medians for the constraint based algorithms are a bit lower. This could, however, be due to the fact that we only did ten runs to decrease the computation time.

	Min	1st Quantile	Median	Mean	3rd Quantile	Max.
Tabu	37.62	37.71	37.78	37.78	37.85	37.92
HC	37.65	37.76	37.79	37.78	37.81	37.87
MMHC	37.51	37.68	37.76	37.75	37.80	37.98
GS	37.57	37.73	37.75	37.76	37.82	37.99
FIAMB	37.57	37.74	37.77	37.77	37.83	37.89
IIAMB	37.59	37.74	37.80	37.81	37.91	37.95

Table 4.10: Percentage of matches predicted correctly for score and hybrid based algorithms

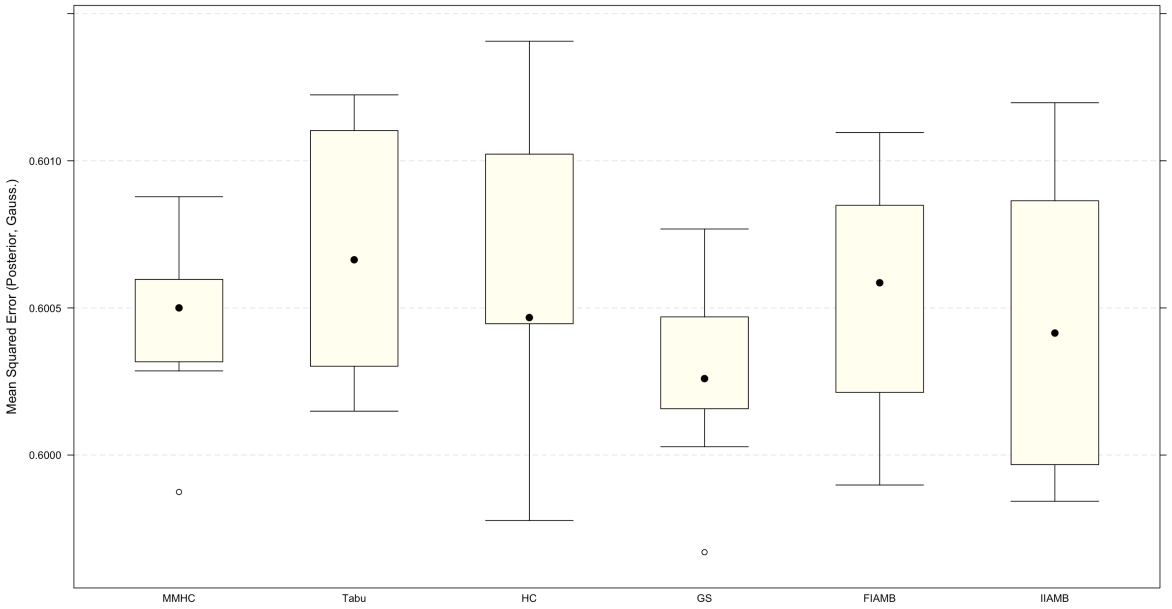


Figure 4.15: Mean square errors for 10 runs for all algorithms with arcs forced towards label

5

Conclusion

In the beginning of this thesis the question was asked whether is was possible to predict football matches with the help of Bayesian Networks and how well this predictions would be. Secondly we also asked ourselves if it was possible to make better predictions than the predictions made by our betting company. To answer these questions we first analyzed the data at hand and modified it in such a way that the variables involved could be used to explain causal relationships in our Bayesian Network.

Unfortunately the answers to the questions posed are not satisfactory as the predictions were often far of the real results. For the first bootstrap method we predicted around 52% correctly which is nowhere close to perfection. However, we saw in our analysis of the data that if we always predict a home win, then we would predict 46% correctly. The cross validation predicted worse than the predictions resulting from the bootstrap method, with only around 37% predicted correctly. This prediction percentage could be explained by the fact that for the cross validation we were predicting only the end result while with the bootstrap method we were predicting the goals of the match. By forcing our structure to contain only arcs towards the nodes we wanted to predict, we did improve our predictions for the bootstrap method. Instead of 52% we now predicted 54.5% correctly. This small increase is maybe due to the fact that some dependencies which were not captured in the first iteration, did have an influence in the end result. Lastly by removing the betting nodes the prediction scores even decreased, probably for the reason that these are important predictors as already seen in the first iteration of the full network. With the cross validation neither of these changes significantly altered the prediction results and the associated errors.

In conclusion we can say that the variables used in these structures are not sufficient enough to make good predictions. To make better predictions it might be wise to include more variables, like more information about the players starting the match or the weather during the day, but this will also significantly increase the computation time needed to learn these structures and its validation methods. We also have to note that the variables involved don't follow the assumptions that were used to model with continuous Bayesian Networks. Although the networks should still be able to make a good approximation when we are not adhering to these assumptions, this might have an influence in the predictions. Lastly, football is a game with a lot of variables involved and it is hard to capture those in a causal model.

A

Table betting companies

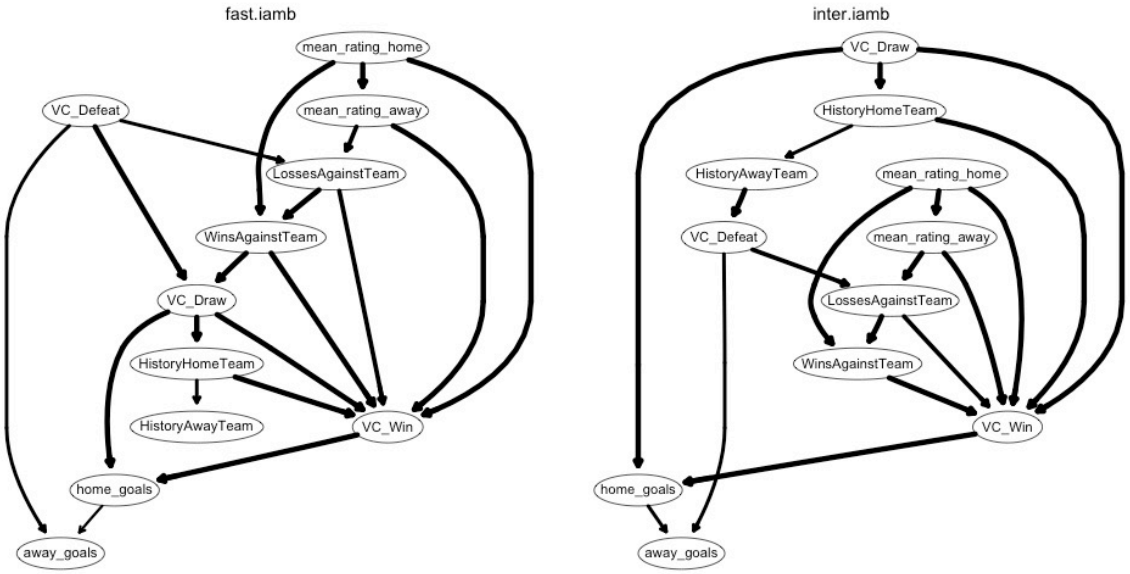
Abbreviation	Better
B365	Bet365
BW	Bet & Win
IW	Interwetten
LB	Ladbrokes
WH	William Hill
SJ	Stan James
VC	VC Bet
GB	Gamebookers
BS	Blue Square
Avg	Average over all betters

Table A.1: Table with betting company names

B

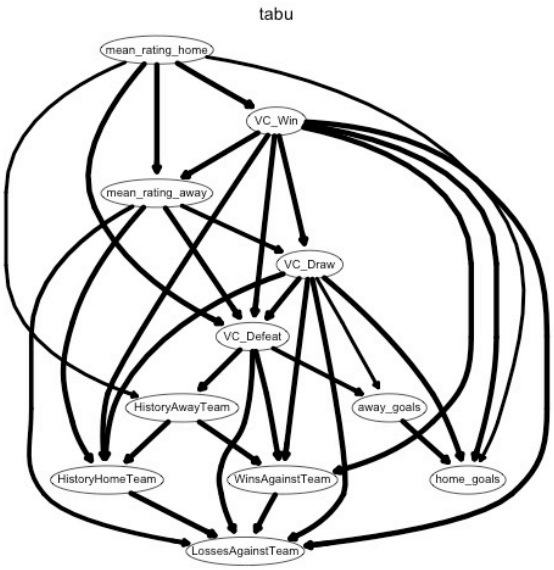
Additional Strength plots

B.1. Simple bootstrap

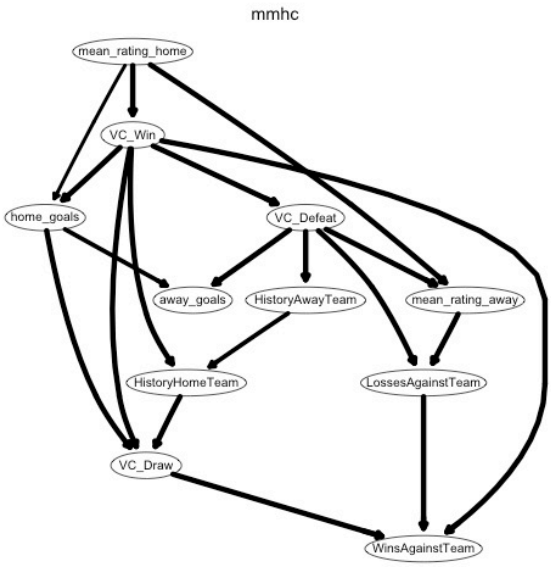


(a) Strength Plot for Fast IAMB

(b) Strength Plot for Inter IAMB

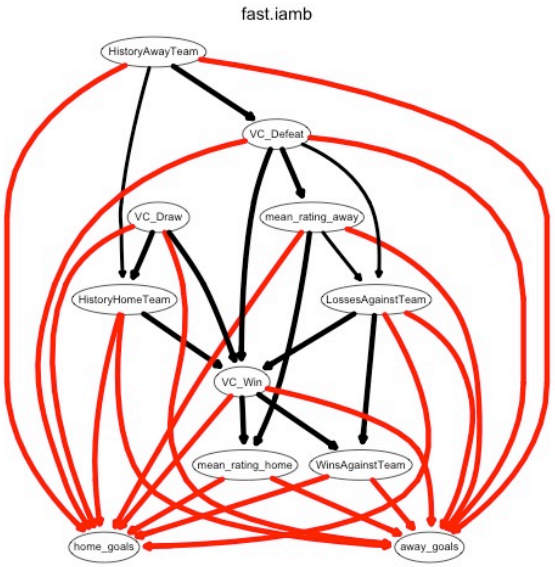


(a) Strength Plot for tabu



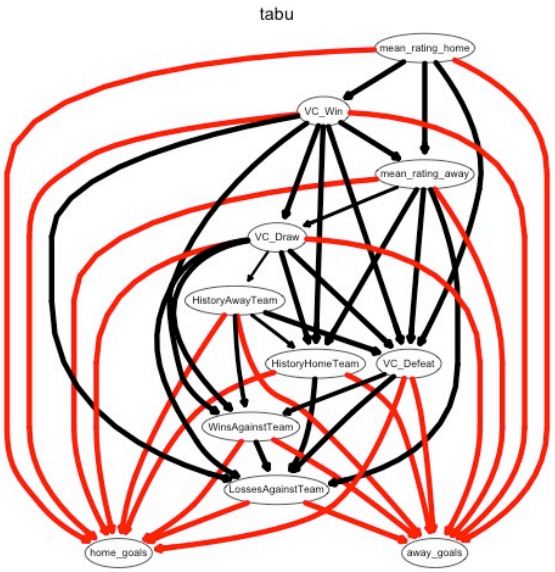
(b) Strength Plot for MMHC

B.2. Bootstrap with all arcs towards goals

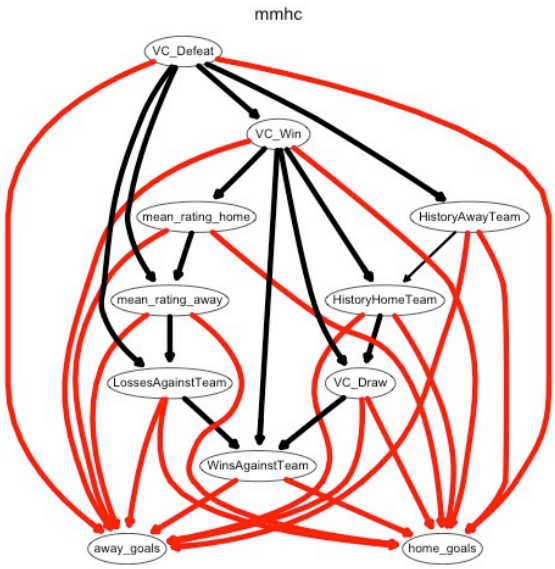


(a) Strength Plot for Fast IAMB with arcs towards goals

(b) Strength Plot for Inter IAMB with arcs towards goals

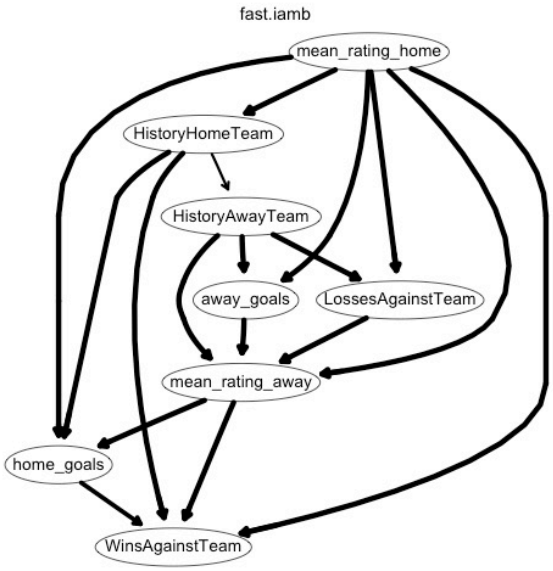


(a) Strength Plot for tabu with arcs towards goals

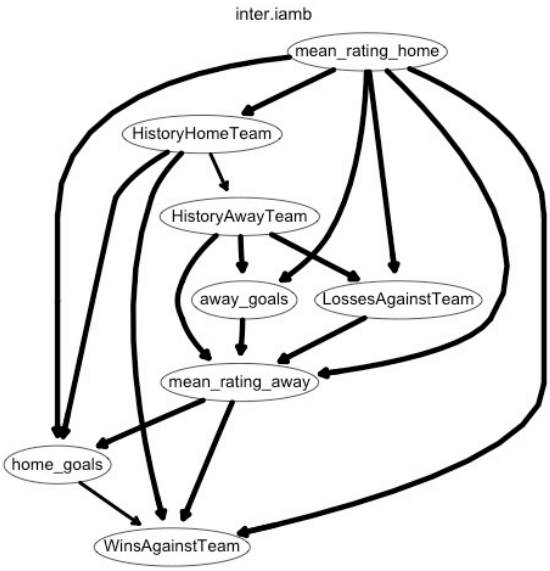


(b) Strength Plot for MMHC with arcs towards goals

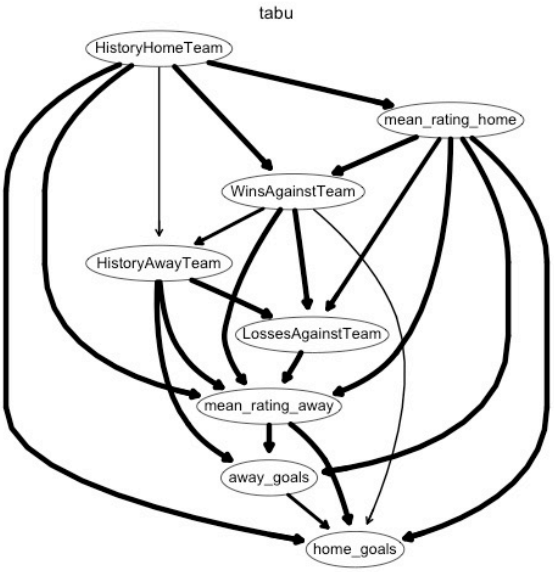
B.3. Bootstrap without betting nodes



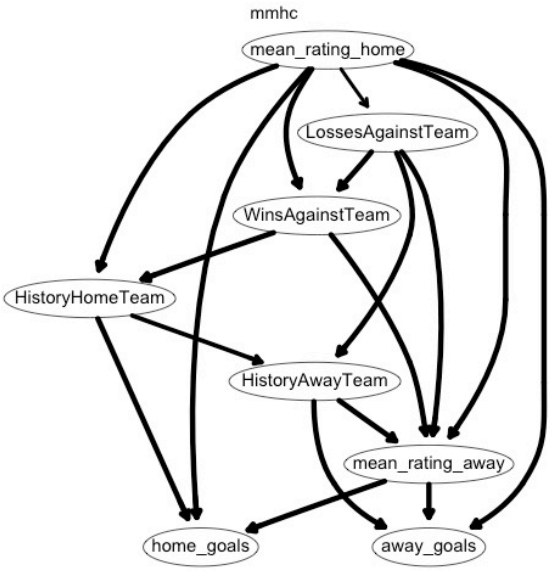
(a) Strength Plot for Fast IAMB without betting nodes



(b) Strength Plot for Inter IAMB without betting nodes



(a) Strength Plot for tabu without betting nodes



(b) Strength Plot for MMHC without betting nodes

C Code

C.1. Python code

C.1.1. Functions

```
#adapted from:
  ↪ https://www.kaggle.com/airback/match-outcome-prediction-in-football
import numpy as np
import pandas as pd
import sqlite3
from time import time
from collections import Counter

def get_match_label(match):
    ''' Derives a label for a given match. '''
    home_goals = match['home_team_goal']
    away_goals = match['away_team_goal']
    label = pd.DataFrame()
    label.loc[0,'match_api_id'] = match['match_api_id']
    if home_goals > away_goals:
        label.loc[0,'label'] = 2
    if home_goals == away_goals:
        label.loc[0,'label'] = 1
    if home_goals < away_goals:
        label.loc[0,'label'] = 0
    return label.loc[0]

def get_fifa_stats(match, player_stats):
    ''' Aggregates fifa stats for a given match. '''
    match_id =  match.match_api_id
    date = match['date']
    players = ['home_player_1', 'home_player_2', 'home_player_3',
      ↪ "home_player_4", "home_player_5",
               "home_player_6", "home_player_7", "home_player_8",
                 ↪ "home_player_9", "home_player_10",
               "home_player_11", "away_player_1", "away_player_2",
                 ↪ "away_player_3", "away_player_4",
               "away_player_5", "away_player_6", "away_player_7",
                 ↪ "away_player_8", "away_player_9",
               "away_player_10", "away_player_11"]
    player_stats_new = pd.DataFrame()
    names = []
```

```
33      for player in players:
34          player_id = match[player]
35          stats = player_stats[player_stats.player_api_id == player_id]
36          current_stats = stats[stats.date < date].sort_values(by = 'date',
              ↪ ascending = False)[:1]
37          if np.isnan(player_id) == True:
38              overall_rating = pd.Series(0)
39          else:
40              current_stats.reset_index(inplace = True, drop = True)
41              overall_rating = pd.Series(current_stats.loc[0,
                  ↪ "overall_rating"])
42          name = "{}_overall_rating".format(player)
43          names.append(name)
44          player_stats_new = pd.concat([player_stats_new, overall_rating],
              ↪ axis = 1)
45      player_stats_new.columns = names
46      player_stats_new['match_api_id'] = match_id
47      #retrieved from:
          ↪ https://www.kaggle.com/yoyocm/how-predict-the-outcome-of-40-matches
48      player_stats_new['mean_rating_home'] =
          ↪ player_stats_new[['home_player_1_overall_rating',
          ↪ 'home_player_2_overall_rating', 'home_player_3_overall_rating',
          ↪ "home_player_4_overall_rating", "home_player_5_overall_rating",
49    "home_player_6_overall_rating", "home_player_7_overall_rating",
        ↪ "home_player_8_overall_rating", "home_player_9_overall_rating",
        ↪ "home_player_10_overall_rating",
50    "home_player_11_overall_rating"]].mean(axis=1)
51      player_stats_new['mean_rating_away'] =
          ↪ player_stats_new[['away_player_1_overall_rating',
          ↪ 'away_player_2_overall_rating', 'away_player_3_overall_rating',
          ↪ "away_player_4_overall_rating", "away_player_5_overall_rating",
52   "away_player_6_overall_rating", "away_player_7_overall_rating",
       ↪ "away_player_8_overall_rating", "away_player_9_overall_rating",
       ↪ "away_player_10_overall_rating",
53   "away_player_11_overall_rating"]].mean(axis=1)
54      player_stats_new.reset_index(inplace = True, drop = True)
55      return player_stats_new.loc[0]
56
57  def get_fifa_data(matches, player_stats, path = None, data_exists =
      ↪ False):
58      ''' Gets fifa data for all matches. '''
59      #Check if fifa data already exists
60      if data_exists == True:
61          fifa_data = pd.read_pickle(path)
62      else:
63          #Apply get_fifa_stats for each match
64          fifa_data = matches.apply(lambda x :get_fifa_stats(x,
              ↪ player_stats), axis = 1)
65      #Return fifa_data
66      return fifa_data
67
68  def get_overall_fifa_rankings(fifa, get_overall = False):
69      ''' Get overall fifa rankings from fifa data. '''
70      temp_data = fifa
71      if get_overall == True:
72          #Get overall stats
```

```
73          data = temp_data.loc[:,(fifa.columns.str.contains('mean_rating'))]
74          data.loc[:,'match_api_id'] = temp_data.loc[:,'match_api_id']
75      else:
76          cols = fifa.loc[:,(fifa.columns.str.contains('date_stat'))]
77          temp_data = fifa.drop(cols.columns, axis = 1)
78          data = temp_data
79      return data
80
81  def get_team_stats(match, team_stats):
82      ''' Aggregates team stats for a given match. '''
83      match_id =  match.match_api_id
84      date = match['date']
85      teams = [”home_team_api_id”, ”away_team_api_id”]
86      team_stats_new = pd.DataFrame()
87      names = []
88      for team in teams:
89          team_id = match[team]
90          stats = team_stats[team_stats.team_api_id == team_id]
91          current_stats = stats[stats.date < date].sort_values(by = 'date',
              ↪ ascending = False)[:1]
92          if current_stats.empty:
93              current_stats = stats[stats.date > date].sort_values(by =
                  ↪ 'date', ascending = False)[:1]
94          if current_stats.empty:
95              a = np.zeros(shape=(1,len(current_stats.columns)))
96              current_stats = pd.DataFrame(a,columns=current_stats.columns)
97          if np.isnan(team_id) == True:
98              buildUpPlaySpeed = pd.Series(0)
99          else:
100             current_stats.reset_index(inplace = True, drop = True)
101             buildUpPlaySpeed = pd.Series(current_stats.loc[0,
                  ↪ ”buildUpPlaySpeed”])
102         name = ”{}_buildUpPlaySpeed”.format(team)
103         names.append(name)
104         team_stats_new = pd.concat([team_stats_new, buildUpPlaySpeed],
              ↪ axis = 1)
105     team_stats_new.columns = names
106     team_stats_new['match_api_id'] = match_id
107     team_stats_new.reset_index(inplace = True, drop = True)
108     return team_stats_new.loc[0]
109
110 def get_team_data(matches, team_stats, path = None, data_exists = False):
111     ''' Gets team data for all matches. '''
112     if data_exists == True:
113         team_data = pd.read_pickle(path)
114     else:
115         team_data = matches.apply(lambda x :get_team_stats(x, team_stats),
              ↪ axis = 1)
116     team_data.columns = [”home_team_buildUpPlaySpeed”,
          ↪ ”away_team_buildUpPlaySpeed”, ”match_api_id”]
117     return team_data
118
119 def get_overall_team_rankings(teamData, get_overall = False):
120     ''' Get overall team rankings from team data. '''
121     temp_data = teamData
122     if get_overall == True:
```

```
123         data =
              ↪ temp_data.loc[:,(teamData.columns.str.contains('buildUpPlaySpeed'))]
124         data.loc[:,'match_api_id'] = temp_data.loc[:,'match_api_id']
125     else:
126         cols =
              ↪ teamData.loc[:,(teamData.columns.str.contains('date_stat'))]
127         temp_data = teamData.drop(cols.columns, axis = 1)
128         data = temp_data
129     data = data[data.home_team_buildUpPlaySpeed != 0]
130     data = data[data.away_team_buildUpPlaySpeed != 0]
131     return data
132
133 def get_last_matches(matches, date, team, x):
134     ''' Get the last x matches of a given team. '''
135     team_matches = matches[(matches['home_team_api_id'] == team) |
          ↪ (matches['away_team_api_id'] == team)]
136     last_matches = team_matches[team_matches.date < date].sort_values(by =
          ↪ 'date', ascending = False).iloc[0:x,:]
137     return last_matches
138
139 def get_last_matches_against_eachother(matches, date, home_team,
      ↪ away_team, x):
140     ''' Get the last x matches of two given teams. '''
141     home_matches = matches[(matches['home_team_api_id'] == home_team) &
          ↪ (matches['away_team_api_id'] == away_team)]
142     away_matches = matches[(matches['home_team_api_id'] == away_team) &
          ↪ (matches['away_team_api_id'] == home_team)]
143     total_matches = pd.concat([home_matches, away_matches])
144     try:
145         last_matches = total_matches[total_matches.date <
              ↪ date].sort_values(by = 'date', ascending = False).iloc[0:x,:]
146     except:
147         last_matches = total_matches[total_matches.date <
              ↪ date].sort_values(by = 'date', ascending =
              ↪ False).iloc[0:total_matches.shape[0],:]
148         if(last_matches.shape[0] > x):
149             print("Error in obtaining matches")
150     return last_matches
151
152 def get_wins(matches, team):
153     ''' Get the number of wins of a specfic team from a set of matches.
          ↪ '''
154     home_wins = int(matches.home_team_goal[(matches.home_team_api_id ==
          ↪ team) & (matches.home_team_goal >
          ↪ matches.away_team_goal)].count())
155     away_wins = int(matches.away_team_goal[(matches.away_team_api_id ==
          ↪ team) & (matches.away_team_goal >
          ↪ matches.home_team_goal)].count())
156     total_wins = home_wins + away_wins
157     g = matches.shape[0]
158     if g == 0:
159         percentage = 0
160     else:
161         percentage = total_wins * 100/ g
162     return percentage
163
```

```
#from:
  ↪ https://www.kaggle.com/hugomathien/squad-visualization-xy-coordinate
def get_setup(match):
    home_players_api_id = list()
    away_players_api_id = list()
    home_players_x = list()
    away_players_x = list()
    home_players_y = list()
    away_players_y = list()
    for i in range(1,12):
        home_players_api_id.append(match['home_player_%d' % i])
        away_players_api_id.append(match['away_player_%d' % i])
        home_players_x.append(match['home_player_X%d' % i])
        away_players_x.append(match['away_player_X%d' % i])
        home_players_y.append(match['home_player_Y%d' % i])
        away_players_y.append(match['away_player_Y%d' % i])
    players_api_id = [home_players_api_id,away_players_api_id]
    players_api_id.append(home_players_api_id) # Home
    players_api_id.append(away_players_api_id) # Away
    home_players_x = [5 if x==1 else x for x in home_players_x]
    away_players_x = [5 if x==1 else x for x in away_players_x]
    players_y = [home_players_y,away_players_y]
    formations = [None] * 2
    for i in range(2):
        formation_dict=Counter(players_y[i]);
        sorted_keys = sorted(formation_dict)
        formation = ''
        for key in sorted_keys[1:-1]:
            y = formation_dict[key]
            formation += '%d-' % y
        formation += '%d' % formation_dict[sorted_keys[-1]]
        formations[i] = formation
    return formations

def get_match_features(match, matches, x):
    ''' Create match specific features for a given match. '''
    date = match.date
    home_team = match.home_team_api_id
    away_team = match.away_team_api_id
    #Get last x matches of home and away team
    matches_home_team = get_last_matches(matches, date, home_team, 5)
    matches_away_team = get_last_matches(matches, date, away_team, 5)
    #Get last x matches of both teams against each other
    last_matches_against = get_last_matches_against_eachother(matches,
      ↪ date, home_team, away_team, 2)
    #Create formation variable
    formation = get_setup(match)
    #Define result data frame
    result = pd.DataFrame()
    #Define ID features
    result.loc[0, 'match_api_id'] = match.match_api_id
    result.loc[0, 'league_id'] = match.league_id
    result.loc[0, 'date'] = match.date
    result.loc[0, 'stage'] = match.stage
    #Create match features
    result.loc[0, 'home_team'] = match.home_team_api_id
```

```
218     result.loc[0, 'away_team'] = match.away_team_api_id
219     result.loc[0, 'home_goals'] = match.home_team_goal
220     result.loc[0, 'away_goals'] = match.away_team_goal
221     result.loc[0, 'games_won_home_team'] = get_wins(matches_home_team,
          ↪ home_team)
222     result.loc[0, 'games_won_away_team'] = get_wins(matches_away_team,
          ↪ away_team)
223     result.loc[0, 'games_against_won'] = get_wins(last_matches_against,
          ↪ home_team)
224     result.loc[0, 'games_against_lost'] = get_wins(last_matches_against,
          ↪ away_team)
225     result.loc[0, 'home_formation'] = formation[0]
226     result.loc[0, 'away_formation'] = formation[1]
227     return result.loc[0]
228
229 def convert_odds_to_prob(match_odds):
230     ''' Converts bookkeeper odds to probabilities. '''
231     match_id = match_odds.loc[:,'match_api_id']
232     bookkeeper = match_odds.loc[:,'bookkeeper']
233     win_odd = match_odds.loc[:,'Win']
234     draw_odd = match_odds.loc[:,'Draw']
235     loss_odd = match_odds.loc[:,'Defeat']
236     win_prob = 1 / win_odd
237     draw_prob = 1 / draw_odd
238     loss_prob = 1 / loss_odd
239     probs = pd.DataFrame()
240     probs.loc[:,'match_api_id'] = match_id
241     probs.loc[:,'bookkeeper'] = bookkeeper
242     probs.loc[:,'Win'] = win_prob
243     probs.loc[:,'Draw'] = draw_prob
244     probs.loc[:,'Defeat'] = loss_prob
245     return probs
246
247 def get_bookkeeper_data(matches, bookkeepers, horizontal = True):
248     ''' Aggregates bookkeeper data for all matches and bookkeepers. '''
249     bk_data = pd.DataFrame()
250     for bookkeeper in bookkeepers:
251         #Find columns containing data of bookkeeper
252         temp_data =
              ↪ matches.loc[:,(matches.columns.str.contains(bookkeeper))]
253         temp_data.loc[:, 'bookkeeper'] = str(bookkeeper)
254         temp_data.loc[:, 'match_api_id'] = matches.loc[:, 'match_api_id']
255         #Rename odds columns and convert to numeric
256         cols = temp_data.columns.values
257         cols[:3] = ['Win','Draw','Defeat']
258         temp_data.columns = cols
259         temp_data.loc[:,'Win'] = pd.to_numeric(temp_data['Win'])
260         temp_data.loc[:,'Draw'] = pd.to_numeric(temp_data['Draw'])
261         temp_data.loc[:,'Defeat'] = pd.to_numeric(temp_data['Defeat'])
262         if(horizontal == True):
263             #Convert data to probs
264             temp_data = convert_odds_to_prob(temp_data)
265             temp_data.drop('match_api_id', axis = 1, inplace = True)
266             temp_data.drop('bookkeeper', axis = 1, inplace = True)
267             win_name = bookkeeper + "_" + "Win"
268             draw_name = bookkeeper + "_" + "Draw"
```

```
269             defeat_name = bookkeeper + "_" + "Defeat"
270             temp_data.columns.values[:3] = [win_name, draw_name,
                  ↪ defeat_name]
271             #Aggregate data
272             bk_data = pd.concat([bk_data, temp_data], axis = 1)
273         else:
274             #Aggregate vertically
275             bk_data = bk_data.append(temp_data, ignore_index = True)
276     #If horizontal add match api id to data
277     if(horizontal == True):
278         temp_data.loc[:, 'match_api_id'] = matches.loc[:, 'match_api_id']
279     return bk_data
280 
281 def get_bookkeeper_probs(matches, bookkeepers, horizontal = False):
282     ''' Get bookkeeper data and convert to probabilities for vertical
          ↪ aggregation. '''
283     data = get_bookkeeper_data(matches, bookkeepers, horizontal = False)
284     probs = convert_odds_to_prob(data)
285     return probs
286 
287 def create_feables(matches, fifa, teamData, bookkeepers, get_overall =
      ↪ False, horizontal = True, x = 10, verbose = True):
288     ''' Create and aggregate features and labels for all matches. '''
289     fifa_stats = get_overall_fifa_rankings(fifa, get_overall)
290     team_stats = get_overall_team_rankings(teamData, get_overall)
291     match_stats = matches.apply(lambda x: get_match_features(x, matches,
          ↪ 10), axis = 1)
292     labels = matches.apply(get_match_label, axis = 1)
293     bk_data = get_bookkeeper_data(matches, bookkeepers, horizontal = True)
294     bk_data.loc[:,'match_api_id'] = matches.loc[:,'match_api_id']
295     features = pd.merge(match_stats, fifa_stats, on = 'match_api_id', how
          ↪ = 'left')
296     features = pd.merge(features, team_stats, on = 'match_api_id', how =
          ↪ 'left')
297     features = pd.merge(features, bk_data, on = 'match_api_id', how =
          ↪ 'left')
298     feables = pd.merge(features, labels, on = 'match_api_id', how =
          ↪ 'left')
299     feables.dropna(inplace = True)
300     return feables
```

C.1.2. Importing

```
1  path = "/.../TW3/BEP/"
2  database = path + 'database.sqlite'
3  conn = sqlite3.connect(database)
4  conn.row_factory = sqlite3.Row
5  cur = conn.cursor()
6  
7  player_data = pd.read_sql("SELECT * FROM Player;", conn)
8  player_stats_data = pd.read_sql("SELECT * FROM Player_Attributes;", conn)
9  match_dataF = pd.read_sql("SELECT * FROM Match;", conn)
10 team_stats_data = pd.read_sql("SELECT * FROM Team_Attributes;", conn)
11 
12 rows = ["country_id", "league_id", "season", "stage", "date",
     ↪ "match_api_id", "home_team_api_id",
```

```
13          "away_team_api_id", "home_team_goal", "away_team_goal",
              ↪ "home_player_1", "home_player_2",
14          "home_player_3", "home_player_4", "home_player_5",
              ↪ "home_player_6", "home_player_7",
15          "home_player_8", "home_player_9", "home_player_10",
              ↪ "home_player_11", "away_player_1",
16          "away_player_2", "away_player_3", "away_player_4",
              ↪ "away_player_5", "away_player_6",
17          "away_player_7", "away_player_8", "away_player_9",
              ↪ "away_player_10", "away_player_11",
18          "home_player_X1", "home_player_X2",
19          "home_player_X3", "home_player_X4", "home_player_X5",
              ↪ "home_player_X6", "home_player_X7",
20          "home_player_X8", "home_player_X9", "home_player_X10",
              ↪ "home_player_X11", "away_player_X1",
21          "away_player_X2", "away_player_X3", "away_player_X4",
              ↪ "away_player_X5", "away_player_X6",
22          "away_player_X7", "away_player_X8", "away_player_X9",
              ↪ "away_player_X10", "away_player_X11",
23          "home_player_Y1", "home_player_Y2",
24          "home_player_Y3", "home_player_Y4", "home_player_Y5",
              ↪ "home_player_Y6", "home_player_Y7",
25          "home_player_Y8", "home_player_Y9", "home_player_Y10",
              ↪ "home_player_Y11", "away_player_Y1",
26          "away_player_Y2", "away_player_Y3", "away_player_Y4",
              ↪ "away_player_Y5", "away_player_Y6",
27          "away_player_Y7", "away_player_Y8", "away_player_Y9",
              ↪ "away_player_Y10", "away_player_Y11",
28          "VCH", "VCA", "VCD"]
29
30  match_dataF.dropna(subset = rows, inplace = True)
31  fifa_data = get_fifa_data(match_data, player_stats_data, data_exists =
      ↪ False)
32  team_data = get_team_data(match_data, team_stats_data, data_exists =
      ↪ False)
33
34  bk_cols_selected = ['VC']
35  feables = create_feables(match_data, fifa_data, team_data,
      ↪ bk_cols_selected, get_overall = True)
36  inputs = feables.drop('match_api_id', axis = 1)
37  inputs = inputs.loc[:,~inputs.columns.str.endswith('_overall_rating')]
38  export_csv = inputs.to_csv (r'/.../TW3/Matches.csv', index = None,
      ↪ header=True)
```

C.2. R code

C.2.1. Best better

```
1  setwd(dirname(parent.frame(2)$ofile))
2  library("bnlearn")
3  library("RSQLite")
4  library("corrplot")
5  library("tidyverse")
6  library("data.table")
7  library("ggplot2")
8  con <- dbConnect(SQLite(), dbname="database.sqlite")
```

```
Match <- tbl_df(dbGetQuery(con,"SELECT * FROM Match JOIN Country on Country.id = Match.country_id
                           JOIN League on League.id = Match.league_id"))
Match3 <- subset(Match, select =-c(goal,shoton,shotoff,foulcommit,card,cross, corner,possession))
setnames(Match3, old = c('name','name..120'), new = c("Country","League"))
Match3$Label <- NA
Match3[Match3$home_team_goal >Match3$away_team_goal,"Label"] <- 2
Match3[Match3$home_team_goal < Match3$away_team_goal,"Label"] <- 1
Match3[Match3$home_team_goal == Match3$away_team_goal,"Label"] <- 0

Match4 <- subset(Match3, select = c(78:107, 113))
Match4 <- subset(Match4, select = -c(PSH, PSD, PSA)) #Removed Pinnacle website because of many NaN's
Match5<- na.omit(Match4)

Match5$AvgW <- NA
Match5$AvgW <- apply(Match5[,seq(1,27, by = 3)], 1, FUN = mean)
Match5$AvgD <- NA
Match5$AvgD <- apply(Match5[,seq(2,27, by = 3)], 1, FUN = mean)
Match5$AvgL <- NA
Match5$AvgL <- apply(Match5[,seq(3,27, by = 3)], 1, FUN = mean)

#
Match5$B365 <- NA
Match5$B365 <- apply(Match5[,c(1:3)], 1, FUN = min)
Match5[Match5$B365 == Match5$B365H,"B365"] <- 2
Match5[Match5$B365 == Match5$B365A,"B365"] <- 1
Match5[Match5$B365 == Match5$B365D,"B365"] <- 0
Match5$B3P <- NA
Match5[Match5$B365 == Match5$Label, "B3P"] <- 1
B3P <- sum(Match5$B3P %in% 1)/nrow(Match5)
#Repeat this for all betting companies

Match5$AvgP <- NA
Match5[Match5$Avg == Match5$Label, "AvgP"] <- 1
AvgP <- sum(Match5$AvgP %in% 1)/nrow(Match5)
```

C.2.2. Functions

```
bootfunc <- function(y, data){
  boot <- boot.strength(data, R = 50, algorithm = y, algorithm.args = list(whitelist = wl1), cluster = cl)
  avg.boot <- averaged.network(boot, threshold = 0.7)
  if (!all(is.na(undirected.arcs(avg.boot)))) {
    avg.boot <- cextend(avg.boot)
  }
  fitted <- bn.fit(avg.boot, data, cluster = cl)
  bootlist <- list("avg" = avg.boot, "b" = boot, "f" = fitted)
  return(bootlist)
}
bncv <- function(data, alg, l){
  lossarg <- list(target = "label")
  cv <- bn.cv(data = data, bn = alg, k = 5, runs = 10, loss = l, loss.args = lossarg, cluster = cl)
  return(cv)
}
```

C.2.3. Main

```
setwd(dirname(parent.frame(2)$ofile))
library("bnlearn")
library("RSQLite")
library("corrplot")
library("tidyverse")
library("data.table")
library("ggplot2")
library("gRain")
library("rbmn")
library("parallel")
library(Metrics)
library(ggplot2)

no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)

matchdata<-read.csv("Matches2.csv", header = TRUE, sep = ',' )
Match <- matchdata
Match <- subset(Match, select = -c(home_formation, away_formation,
  ↪ home_team, away_team))
setnames(Match, old = c('games_against_won', 'games_won_home_team',
  ↪ 'away_team_buildUpPlaySpeed',
                        'home_team_buildUpPlaySpeed',
                          ↪ 'games_won_away_team', 'games_against_lost'),
         new = c('WinsAgainstTeam', 'HistoryHomeTeam', 'ATPlaySpeed',
           ↪ 'HTPlaySpeed', 'HistoryAwayTeam', 'LossesAgainstTeam'))

Match$VC <- NA
Match$VC <- apply(Match[,c("VC_Win", "VC_Draw", "VC_Defeat")], 1, FUN =
  ↪ max)
Match[Match$VC == Match$VC_Win,"VC"] <- 2
Match[Match$VC == Match$VC_Draw,"VC"] <- 1
Match[Match$VC == Match$VC_Defeat,"VC"] <- 0

Match$VCP <- NA
Match[Match$VC == Match$label, "VCP"] <- 1
VCP <- sum(Match$VCP %in% 1)/nrow(Match)
Match$VC_Win <- Match$VC_Win*100
Match$VC_Draw <- Match$VC_Draw*100
Match$VC_Defeat <- Match$VC_Defeat*100
Match <- subset(Match, select = -c(VC, VCP, HTPlaySpeed, ATPlaySpeed))
Match <- subset(Match, select = -c(date, day, league_id, day_number, year,
  ↪ stage))
Network <- lapply(Match, as.numeric)
n <- length(Network[[1]])
Network <- structure(Network, row.names = c(NA, -n), class = "data.frame")
Network2 <- subset(Network, select = -c(label))

training <- Network2 %>% sample_frac(.8)
test <- setdiff(Network2, training)

fullNetwork.hc <-hc(Network)
graphviz.plot(fullNetwork.hc, shape = "ellipse", main = "hc")
fullNetwork.iamb <- iamb(Network)
fullNetwork.tabu <- tabu(Network)
```

```
fullNetwork.gs <- gs(Network)
fullNetwork.inter.iamb <- inter.iamb(Network)
fullNetwork.fast.iamb <- fast.iamb(Network)
fullNetwork.mmhc <- mmhc(Network)
par(mfrow = c(1,2))
graphviz.compare(fullNetwork.gs, fullNetwork.fast.iamb, fullNetwork.iamb,
  ↪ fullNetwork.inter.iamb, shape = "ellipse", main = c("GS", "FIAMB",
  ↪ "IAMB", "IIAMB"))
par(mfrow=c(1,1))
rho <- cor(Network)
corrplot(rho, method = "circle", order = "hclust")

wl1 <- matrix(c(names(Network)[c(3:11)], rep("label", 9)), ncol = 2)
bl2 <- data.frame(from = c("home_goals", "away_goals"), to =
  ↪ c("away_goals", "home_goals"))
wl2 <- matrix(c(names(Network)[c(3:8)], rep("home_goals", 6)), ncol = 2)
wl3 <- matrix(c(names(Network)[c(3:8)], rep("away_goals", 6)), ncol = 2)
wlf <- rbind(wl2, wl3)

###Bootstrap
algs = c("hc", "gs", "fast.iamb", "inter.iamb", "tabu", "mmhc")
par(mfrow=c(1,1))
for (alg in algs){
  bf <- bootfunc(alg, training)
  predresH <- predict(bf$f, c("home_goals"), test, method = "bayes-lw")
  predresA <- predict(bf$f, c("away_goals"), test, method = "bayes-lw")
  testresults <- data.frame(predresH, test$home_goals, predresA,
    ↪ test$away_goals)
  colnames(testresults)<-c("PredictedHomeGoals", "Home_goals",
    ↪ "PredictedAwayGoals", "Away_goals")
  testresults$PredictedLabel <- NA
  testresults[testresults$PredictedHomeGoals >
    ↪ testresults$PredictedAwayGoals,"PredictedLabel"] <- 2
  testresults[testresults$PredictedHomeGoals ==
    ↪ testresults$PredictedAwayGoals,"PredictedLabel"] <- 1
  testresults[testresults$PredictedHomeGoals <
    ↪ testresults$PredictedAwayGoals,"PredictedLabel"] <- 0

  testresults$RealLabel <- NA
  testresults[testresults$Home_goals > testresults$Away_goals,
    ↪ "RealLabel"] <- 2
  testresults[testresults$Home_goals == testresults$Away_goals,
    ↪ "RealLabel"] <- 1
  testresults[testresults$Home_goals < testresults$Away_goals,
    ↪ "RealLabel"] <- 0

  sp <- paste("StrengthPlot_", alg, ".jpeg", sep="")
  jpeg(sp)
  strength.plot(bf$avg, bf$b, shape = "ellipse", main = alg)
  dev.off()
}

##BNCV
algs = c("hc", "tabu", "mmhc")
Network3 <- subset(Network, select = -c(home_goals, away_goals))
for (alg in algs){
```

```
95    bf <- bootfunc(alg, Network3)
96    vals <- bncv(Network3, bf$avg, "mse-lw")
97    err = numeric(10)
98    for (i in 1:10) {
99      tt = table(round(unlist(sapply(vals[[i]], '[[',
          ↪ "predicted"))),unlist(sapply(vals[[i]], '[[', "observed")))
100     err[i] = sum(diag(tt)) / sum(tt) * 100
101   }
102   print(alg)
103   print(summary(err))
104 }
105 plot(bncv_mmhc, bncv_tabu, bncv_hc, bncv_gs, bncv_fast.iamb,
    ↪ bncv_inter.iamb, xlab = c("MMHC", "Tabu", "HC", "GS", "FIAMB",
    ↪ "IIAMB"), connect = FALSE)
106 plot(bncv_mmhc, bncv_tabu,  bncv_hc, xlab = c("MMHC", "Tabu", "HC"))
```

Bibliography

[1] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2007.

[2] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

[3] Richard E. Neapolitan. *Learning Bayesian networks*. Pearson Prentice Hall, 2004.

[4] Marco Scutari. Learning bayesian networks with the bnlearn r package. *Journal of Statistical Software, Articles*, 35(3):1–22, 2010. ISSN 1548-7660. doi: 10.18637/jss.v035.i03. URL https://www.jstatsoft.org/v035/i03.

[5] Marco Scutari and Jean-Baptiste Denis. *Bayesian networks: with examples in R*. CRC Press, Taylor & Francis Group, 2015.

[6] Raheel Shaikh. Cross validation explained: Evaluating estimator performance., Nov 2018. URL https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85.