L-Università ta' Malta
Faculty of Science

Department of
Statistics &
Operations Research

Using Bayesian Networks and other Classification Techniques to Classify Potential Problem Gamblers.

Christian Anthony Farrugia

May 2022

Supervisor: Dr. Mark Anthony Caruana

A dissertation presented by the Faculty of Science in part fulfilment of the requirements for the degree of Bachelor of Science (Hons.) at the University of Malta.

Abstract

Christian Anthony Farrugia, B.Sc (Hons)

Department of Statistics & Operations Research, May 2022

University of Malta

The grave consequences suffered by online problem gamblers has led to a growing interest in responsible gambling measures with the intention of preventing players from reaching such a vulnerable state. The focus of this dissertation is to apply statistical learning and machine learning techniques to predict whether a player is most likely a problem gambler or not whilst identifying which variables were deemed useful predictors of problem gambling. As a proxy measure for problem gambling we will use self-exclusion. Bayesian networks, random forests, boosting and logistic regression are implemented on a data set containing historical data obtained from a local medium-sized Malta Gambling Authority (MGA) gambling operator. The models will be compared based on goodness of fit measures such as the predictive accuracy and Area Under the Curve (AUC). Across almost all measures, the random forest model was superior to the rest of the models, whilst the Bayesian Networks performed better than logistic regression in terms of predictive accuracy.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr Mark Anthony Caruana without whose constant support and advice this dissertation would not have been possible. I would also like to express my sincere appreciation towards all the lecturers within the Department of Statistics and Operations Research for their excellent teaching and guidance throughout the past four years.

Lastly, I would like to thank my family and friends who were pillars of support throughout this academic journey.

To my parents Sergio and Maria and my sister Bernice.

Contents

List of Figures

List of Tables

List of Symbols

$\mathbf{X}$	Random vector of length d
$\mathcal{D}$	Data set containing n independent observations of $\mathbf{X}$
n	total number of observations in the data set
C	Binary class variable
$\mathcal{Z}$	Data set containing n independent observations of $(\mathbf{X}, C)$
$\mathcal{X}$	Sample space for $\mathbf{X}$
$\mathcal{X}_j$	Sample space for variable X_j
Π_α	Parent set of variable X_j
$\mathcal{G}$	DAG
$\mathbf{H}(\cdot)$	entropy function
$\mathbf{D}(\cdot \parallel \cdot)$	relative entropy function
$\Delta(\cdot, \cdot)$	decrease in impurity
$h(\cdot, \cdot)$	decision tree classifier

List of Abbreviations

CL	Chow-Liu
FSSJ	Forward Sequential Selection and Joining
BN	Bayesian Network
SVM	Support Vector Machine
ANN	Artificial Neural Network
BNN	Bayesian Neural Network
DAG	Directed Acyclic Graph
NB	Naïve Bayes
CPD	Conditional Probability Distribution
CART	Classification and Regression Trees
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
ROC	Receiver Operating Characteristic
AUC	Area Under Curve

Chapter 1

Introduction

1.1 Problem Gambling and Machine Learning

The term problem gambling refers to all betting behaviour patterns which may disrupt or compromise the personal life of a gambler. Such behaviour can lead to grave consequences suffered by the staker, such as, but not limited to, debt, failed relationships and loss of employment. In particular, Petry [28] found that online gambling is associated with inferior mental and physical health. On the other hand, a company having a large amount of problem gamblers in their clientele may enjoy higher revenues but risk incurring hefty fines or the loss of their gambling licenses. The latter clash of interests has prompted betting companies to investigate different approaches that seem viable when dealing with problem gamblers. We cover some of these investigations in chapter 2.

Nowadays, gambling is more accessible than ever. An individual who is keen to gamble an amount of money does not need to find a physical casino to wager his money at but needs simply to log on the internet and make an account with one of the many online gambling sites in the market. The ease of access to gambling through the internet has brought about a vast body of literature which aims to study the impact online gambling has on individuals when compared to traditional (offline) gambling. A study conducted by Griffiths et al. [15] suggests that online gambling is more likely to lead to problem gambling than offline gambling. McCormack &

Griffiths [22] conducted a survey on a sample of both online and offline gamblers and found that the players perceived online gambling to be more addictive. Furthermore, convenience and anonymity were both identified as major factors contributing to the players' inclination towards online gambling. In a comparison of online vs offline gamblers conducted by Hubert & Griffiths [17], suicidal thoughts and signs of depression were observed predominantly in the offline gamblers.

Since online gambling has led to an abundance of data on players' transactions on betting sites, statistical and machine learning techniques are frequently used to analyze problem gambling as we will see in chapter 2. Statistical learning and machine learning consist of an algorithm which is trained on a data set to be able to detect patterns or form predictions. These can be further subdivided in two subgroups, namely supervised and unsupervised learning. The former consists of a data set where the variable of interest, often called the dependent/response variable, is labelled. In this study, as will be explained in more detail in chapter 5, the observations in the data set will be labelled according to whether they correspond to a user who is a problem gambler or not, thus we will be dealing with supervised learning. Contrarily, in unsupervised learning the observations are unlabelled and the intention of the model is to detect patterns in the data. Such a technique may be used to group a sample of gamblers into different clusters based on similarities in their transactional data. This study will not delve into any unsupervised learning techniques.

Supervised learning is further divided into two categories, namely classification and regression. In classification, the dependent variable is categorical and the aim is to predict the category of a given observation. On the other hand, in regression problems the dependent variable is continuous and the aim is to estimate the response value of an observation. Regression techniques will not be covered in this dissertation. However, we will be covering classification techniques, namely Bayesian networks, random forests and logistic regression.

Using statistical and machine learning models to detect problem gambling can be useful for both the gamblers and the gambling operator. By preempting that a user is likely to become a problem gambler, the betting operator may opt to send responsible gaming messages to the gambler in an attempt to increase their awareness on the amount of time or money they have spent on the site. This can alleviate the gambler's potential increase in socio-economic problems while diminishing the risk of the company incurring a fine due to lack of adherence to responsible gambling regulations.

1.2 Structure of the Dissertation

The general structure of the dissertation will be outlined in this section. Chapter 2 discusses past studies which leveraged machine learning techniques in order to predict problem gambling. The techniques involved are varied and include, but are not limited to, Bayesian networks, random forests and neural networks. We will also mention studies where Bayesian networks were applied to domains other than gambling.

Chapter 3 is where we outline and explain theoretical results regarding Bayesian networks. We first introduce the concept of a BN and how its structure can be built from the given data using either the Chow-Liu(CL) or the FSSJ algorithm. Consequently, tree-based methods are introduced in chapter 4. The latter chapter is initiated by discussing how decision trees can be used for classification problems and how these are linked to random forests and boosting.

In chapter 5 we discuss how the theory from chapters 3 and 4 was applied to our data set using R software. In order to analyze and compare the models, various goodness of fit measures pertaining to each model will be reported. Measures of predictive ability will also be compared across the various models. In the final chapter, the theory and results of the dissertation are outlined. Moreover, limitations of the study are discussed and suggestions for further improvement are made.

Chapter 2

Literature Review

In this chapter we highlight the different attempts that have been done to predict problem gambling using machine learning techniques. Specifically, the methodologies of these studies will be briefly outlined and comments will be made on the model diagnostics obtained. Moreover, at times we outline the variables considered by the researcher in consideration to be useful for the prediction of problem gambling. Apart from mentioning studies where Bayesian networks were used in the online gambling domain, we shall mention some of their applications in other domains such as medicine and finance.

In a study by Akhter [3], a dataset containing gambling transactions and other variables of over 30000 users of an online gambling site is analysed. The company predefined a set of conditions and considered a user who satisfied any one of these conditions to be a problem gambler. Examples of such conditions were whether the player had self excluded or not or whether the user had spent more than a specific number of hours active on the site. Thus the data set labelled players as problem gamblers or not depending on whether any of these conditions were satisfied. Using this labelled data, the author fitted a Naïve Bayes model as well as linear SVM and found the latter to have higher accuracy, precision and recall. The accuracy of the linear SVM algorithm was 98.34% whilst that of the Naïve Bayes was 96.34%. A similar approach was adopted by Sikiric [32] who fitted Bayesian networks to a data set of previously identified problem gamblers provided by the company ATG.

In this study, the 1531 players were separated into seven risk groups according to variables such as gender, age and whether the player had any deposit limits. The model achieved an accuracy of 94% in predicting the risk group of each user.

Since we cannot know for certain whether a player is a problem gambler or not just by looking at their transactions and/or past data, past studies have considered self-exclusion of a user from a gambling site to be a strong indicator of problem gambling. An example of a study which adopted such an approach is that made by Philander [29]. In his study, nine supervised learning techniques were evaluated on a data set which labelled players as problem gamblers if they had self-excluded in the past. Out of these nine techniques (namely Step-wise Logistic regression, GLM Lasso, Neural Network regression, Neural Network classification, SVM regression, SVM C-classification, SVM one class classification, Random forest regression and Random forest Classification) Artificial Neural Networks were found to be the most effective in predicting self-exclusion, but still failed to classify a large number of potential problem gamblers. The author proposed that the addition of behavioural variables apart from trajectory (total amount of money wagered), frequency (days active), intensity (how regularly the gambler places bets on active days), and variability (the standard deviation of the amount of money gambled) would yield to more accurate prediction of problem gambling. The author referred to a study made by Adami et al. [1] which identified such a variable, namely *sawtooth*. The sawtooth variable flags sudden surges or declines in a player's wagered amount. The author explains that a rapid drop in money wagered may signify that the gambler has exhausted his/her funds and will thus be inactive for a while in order to accumulate more money to be able to continue with their usual betting activity.

Percy et al. [26] argued in favour of using self-exclusion as a proxy for problem-gambling and therefore applied this technique on a data set provided by an online gambling operator, namely IGT. The data set consisted of 845 online gamblers and the variables observed were the trajectory,frequency, intensity,variability as well as the session time (the total amount of time a player spends on the site). The ma-

chine learning techniques of choice that were applied were random forests, logistic regression, Bayesian networks and neural networks. The random forest was identified as the most accurate predictor of self-exclusion and the Bayesian network was the second best performing method. The authors interestingly pointed out that the Bayesian network was superior in identifying players who did not self-exclude than identfying players who did self-exclude. Another study which used the same proxy for problem gambling was that carried out by Bermell [7] in his study which consisted of a data set with 916,312 online casino customers, 75,979 of which had self-excluded in the past and were thus labelled as problem gamblers. In his work, Bermell found that recurrent neural networks performed substantially better than the Gradient Boosting technique which was being used by the company from which he obtained the data set. Buttigieg [9] also implemented the Neural Network framework, specifically Bayesian neural networks (BNN) and artificial neural networks(ANN). An online gaming company provided the author with a data set consisting of 30,706 observations which were labelled depending on whether the users were considered problem gamblers or not by the company. On the data set provied, ANN performed better than the BNN in terms of accuracy, however the BNN had a superior recall metric, meaning that the BNN model is superior to its ANN counterpart if the aim of the researcher or the company is to reduce the number of players who are falsely predicted as being non-problematic gamblers.

Auer & Griffiths [4] argue that setting gambling limits significantly reduces the amount of money wagered by a gambler and hence suggest that influencing the players' limit-setting behavior can lead to more responsible gambling. After evaluating various machine learning algorithms, the Gradient Boosting technique was found to perform best on the test data set. The authors additionally noted that one of the most influential variables was the total amount of money wagered.

An investigation carried out by Quilty et al. [30] aimed towards identifying behavioural characteristics which indicate signs of problem gambling. Their study was conducted using a sample of 503 participants, 275 of which were psychiatric outpa-

tients and 228 where community gamblers. An interesting outcome of this study was that by using logistic regression on the variable indicating overall gambling frequency, fairly well results were obtained, indicating the high influence that a user's gambling frequency has on their likelihood of being a problematic gambler.

As one may anticipate, Bayesian networks have applications that stretch beyond the gaming industry. Agrahari et al. [2] used Bayesian networks on gene expression data to classify two types of hematological malignancies, namely, acute myeloid leukemia (AML) (a cancer of the myeloid blood cells) and myelodysplastic syndrome (MDS) (a disease that affects myeloid cells in the bone marrow and the blood). Hematological malignancies are primary cancers of the blood and blood-forming organs. Another study from the same domain was conducted by Gevaert et al. [14] who used Bayesian networks to predict the prognosis of breast cancer and managed to obtain a model with an AUC of 0.85.

An interesting research carried out by Spyroglou et al. [33] applied Bayesian networks in an attempt to predict asthma exacerbation after medication discontinuation. Note that to obtain optimal control of the disease, the medicine should be gradually reduced and then stopped. The aim of the study was to get an indication of whether the gradual discontinuation of asthma medication would lead to the patient's asthma condition improving or aggravating. The proposed Bayesian network was fitted on a data set containing repeated measurements from 65 patients. The classifier managed to predict whether a patient's disease will aggravate with an accuracy of 93.84% and a sensitivity of 90.9%. The authors added that the structure of the Bayesian network obtained may help clinicians when assessing whether to reduce the asthma medication given to a patient or not.

Bayesian networks are also linked with engineering, as is evident in a study by Mansour et al. [21] which applies Bayesian networks for fault diagnosis of electric components in a power station. The authors discuss how the use of Bayesian networks effectuates real-time fault diagnosis which is indispensable for the operator of the control room of the power station. Bayesian networks were implemented in a study by Triki & Boujelbene [34] to evaluate credit risk of clients who request a

bank loan. The final structure of the Bayesian network fitted on a data set of 6240 customers indicated that age and gender are significant predictors of a loan default. Another application of Bayesian networks in the domain of finance is presented in a study of Demirer et al. [12]. The authors propose the use of Bayesian networks to assist in portfolio decision making. In their study, both macro-economic variables such as inflation and firm specific features such as its earnings are used in the structure learning of the Bayesian network. It is portrayed that the latter can then be used to predict the price increase or decline of a stock, which is of course of great use when deciding whether to invest in a company or not.

The literature we have discussed above suggests that using machine learning techniques can yield promising results when attempting to predict problem gambling. Additionally, previous research we have discussed above advocate for the potency of Bayesian networks and random forests for classification purposes in different domains, especially in online gambling. In our study, we will be using some of the variables mentioned above, such as frequency and trajectory, as well as other features which are mentioned and described in section 5.1. As was done in previous research discussed above, we will be using self-exclusion as a proxy for problem gambling.

Chapter 3

Bayesian Networks

3.1 Introduction

A Bayesian network (BN) is a type of probabilistic model represented by a directed acyclic graph, where the nodes represent the variables of interest. In the case of our data set, examples of such variables of interest are the total deposited amount and total money wagered by a gambler. A BN can capture the conditional dependencies or lack thereof between variables. If a variable is believed to influence another variable, then the nodes corresponding to said variables will be connected by a directed edge.

The material presented in this chapter was adapted from a number of books, predominantly from the work of Koski Noble [19] and Friedman [18]. Apart from defining what a Bayesian network is, in this chapter we will see how to build the best fitting BN model from a given data set. This is called structure learning of a BN (see section 3.5). First, we define the Chow-Liu (CL) algorithm and explain how it is used to build the structure of a BN which obeys certain stringent assumptions (see subsection 3.5.1). We then demonstrate in detail that the CL algorithm finds the graph structure which maximises the likelihood of the data in order to estimate the joint probability distribution which best explains the observed data. Moreover, we will cover Kruskal's algorithm which plays a major role in the CL algorithm and is used to find the maximum weight spanning tree. We then compare the latter to the

FSSJ algorithm [5] which as we will see, is a heuristic procedure which relaxes the assumptions made by the CL algorithm on the possible graph structure the BM can have. As will be explained in section 3.5.2, the FSSJ algorithm has the advantage of performing feature selection by only including a subset of the available predictors in the final model. This is particularly useful in cases where we have a relatively large amount of variables of interest and we want to choose the best fitting model with the optimal amount of variables. After the structure of the BN for our data has been established, we discuss how to perform parameter estimation. The parameters of interest are conditional probability distributions (CPDs) that will be defined and explained in section 3.5. The aim of these parameters is to provide a useful tool for predicting the value of the target variable based on its observed variables of interest.

3.2 Notation and Definitions

In this section we shall introduce some definitions which will be useful in later sections, as well as clarify the notation which will be used. Let n denote the sample size of our data set and d the number of predictor variables in the data set. The latter random variables are all categorical and will be denoted by $X_1, ..., X_d$. We will let $\mathcal{D} = \left(\boldsymbol{x}_{(1)}^T, ..., \boldsymbol{x}_{(n)}^T\right)$ be a data set of n independent observations of the random vector $\boldsymbol{X} = (X_1, ..., X_d)$. On the other hand, $\mathcal{Z} = \left((\boldsymbol{x}_{(1)}, c_{(1)})^T, ..., (\boldsymbol{x}_{(n)}, c_{(n)})^T\right)$ will be used to denote a data set containing n independent observations of $\boldsymbol{X}$ and the binary random variable C, taking values in either $\{0, 1\}$ or in the set $\{-1, 1\}$ depending on the application in consideration.

Moreover, we denote the state space for variable X_j as $\mathcal{X}_j = \{x_j^{(1)}, ..., x_j^{(k_j)}\}$, i.e x_j can take k_j possible values. The state space $\mathcal{X}_j$ of X_j is effectively the range of values that X_j can take. For example, suppose X_j is a discrete random variable denoting the country of birth of a player. For the purpose of this example, assume that the country can either be Sweden, Denmark or Finland, then $\mathcal{X}_j = \{x_j^{(1)}, x_j^{(2)}, x_j^{(3)}\} = \{Sweden, Denmark, Finland\}$.

Hence the state space for $\mathbf{X}$ is given by

$$\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_d = \{(x_1^{(i_1)}, ..., x_d^{(i_d)}) |\ i_1 \in \{1, ..., k_1\}, ..., i_d \in \{1, ..., k_d\}\}.$$

Thus an observed vector, also known as an instantiation of $\mathbf{X}$, is of the form $\mathbf{x} = (x_1^{(i_1)}, ..., x_d^{(i_d)})$.

At times, we shall write $P(\mathbf{X} = \mathbf{x})$ as $P_{\mathbf{X}}(\mathbf{x})$. In certain instances, the latter may also be written as $P(\mathbf{x})$ to ease the complexity of notation.

By adapting the definitions given by Friedman et al. [18], we first define the Empirical PDF for the univariate case in definition 3.2.1 and extend it to the multivariate case in 3.2.2.

Definition 3.2.1. (Empirical PDF - Univariate case)
Let X be a discrete random variable with PDF P and let $X_1, ..., X_n$ be n independent observations of X. The empirical estimate of P, denoted by $\hat{P}$ is given by

$$\hat{P}(X = j) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{X_i = j\}}.$$

This may be easily generalized for a random vector $\boldsymbol{X} = (X_1, ..., X_d)$ where each X_i is a discrete random variable with finite state space $\mathcal{X}_i$. For $i = 1, ..., d$, let $x_i \in \mathcal{X}_i$.

Definition 3.2.2. (Empirical PDF - Multivariate case) Let $\boldsymbol{X}$ be as defined above with PDF P. Moreover, let $\mathcal{D} = (\boldsymbol{x}_{(1)}, ..., \boldsymbol{x}_{(n)})$ be a data set containing n independent observations of $\boldsymbol{X}$. Then the empirical estimate $\hat{P}$ of P is given by

$$\hat{P}(X_1 = x_1, ..., X_d = x_d) = \frac{1}{n} \sum_{j=1}^{n} \mathbb{1}_{\{\boldsymbol{x}_{(j)} = (x_1, ..., x_d)\}}.$$

The identity proved by the following lemma will be used repeatedly throughout this chapter.

Lemma 3.2.1. (Chain rule of probability) For a collection of random variables $X_1, ..., X_d$,

$$P(X_1, ..., X_d) = \prod_{i=1}^{d} P(X_i | X_{i-1}, ..., X_1)$$

Proof. Recall that for two random events A and B, $P(A \cap B) = P(B|A)P(A)$. Thus by repeatedly applying this argument, we get that

$$
\begin{aligned}
P(X_1, ..., X_d) &= P(X_d|X_{i-1}, ..., X_1)P(X_1, ..., X_{d-1}) \\
&= P((X_d|X_{i-1}, ..., X_1))P(X_{d-1}|X_1, ..., X_{d-2})P(X_1, ..., X_{d-2}) \\
&= ... \\
&= P((X_d|X_{i-1}, ..., X_1))P(X_{d-1}|X_1, ..., X_{d-2})P(X_1, ..., X_{d-2})...P(X_2|X_1)P(X_1) \\
&= \prod_{i=1}^{d} P(X_i|X_{i-1}, ..., X_1)
\end{aligned}
$$

□

A graph will be denoted by $\mathcal{G}$, its directed edge set by D and its vertices by V. In the context of Bayesian networks, each node represents a random variable. Moreover, in the case where the graph has undirected edges, we shall denote the set of such edges by E. Definitions 3.2.3 and 3.3.1 below are both given by Koski et al. [19]. These definitions will be used in the context of Bayesian networks, specifically in definition 3.3.1.

Definition 3.2.3. (Parent)

Consider a directed graph $\mathcal{G} = (V, D)$. For any node $X_j \in V$, the *parent set* of X_j is defined as

$$\Pi_j = \{X_k \in V | (X_k, X_j) \in D\}.$$

We shall denote the indexing set of Π_j by $s(j)$. Therefore if the parent set of a variable X_j is $\Pi_j = \{X_1, X_2, X_3\}$, then $s(j) = \{1, 2, 3\}$. If $\Pi_j = \emptyset$ then we let $s(j) = 0$.

As we will see later on in this chapter, Bayesian networks are represented graphically by a certain type of graphs called Directed Acyclic Graphs. We define the latter below.

Definition 3.2.4. (Directed Acyclic Graph (DAG)) A graph $\mathcal{G} = (V, E)$ is said to be a directed acyclic graph if each edge is directed and for any node $\alpha \in V$ there does not exist any set of distinct nodes $\tau_1, ..., \tau_m \in V$ such that $\alpha \neq \tau_i$ for all $i = 1, ..., m$

and $(\alpha, \tau_1, ..., \tau_m, \alpha)$ forms a directed path. This means that there are no directed m-cycles in $\mathcal{G}$ for $m \geq 1$.

Definition 3.2.5. (Descendants) Let $\mathcal{G} = (V, D)$ be a DAG with vertices $V = \{X_1, ..., X_d\}$. The variable X_k is a descendant of X_j if there exists a directed path from X_j to X_k in $\mathcal{G}$. The set of descendants of a variable X_j is denoted by V_j.

3.3 Bayesian Networks

In this section we will first define what a Bayesian network is in definition 3.3.1. The rest of the section will be dedicated to giving clear explanations on how to interpret BNs in order to give logical explanations related to the variables in consideration. Examples related to real world domains, predominantly online gambling, will be discussed throughout the section with the aim of enhancing our understanding.

Definition 3.3.1. A Bayesian network is a pair $(\mathcal{G}, P)$ where $\mathcal{G} = (V, D)$ is a DAG with node set $V = \{X_1, ..., X_d\}$ for some $d \in \mathbb{N}$, D is the directed edge set and P is a joint probability density function indexed by a parameter space $\mathbf{\Psi}$ over d discrete random variables, $X_1, ..., X_d$. Let $\mathcal{X}$ denote the finite state space of $\mathbf{X} = (X_1, ..., X_d)$. The pair$(\mathcal{G}, P)$ satisfies the following conditions:

- For each $\boldsymbol{\psi} \in \mathbf{\Psi}$, $P_{X_1,...,X_d|\boldsymbol{\psi}}(x_1, ..., x_d|\boldsymbol{\psi}) : \mathcal{X} \to [0, 1]$ and $\sum_{\mathbf{x}\in\mathcal{X}} P(\boldsymbol{x}|\boldsymbol{\psi}) = 1$.
- For each node $X_v \in V$ with no parent variables, there is assigned a probability distribution P_{X_v}. To each variable X_v with a non-empty parent set Π_v, we associate the conditional probability distribution $P_{X_v|\Pi_v}$.
- The joint probability function
  $P_{X_1,...,X_d} = P_{X_1} \prod_{i=2}^{d} P_{X_i|X_1,...,X_{i-1}}$ may be factorized as
  $P_{X_1,...,X_d} = \prod_{v=1}^{d} P_{X_v|\Pi_v}$. We say that P factorizes along $\mathcal{G}$.

We now introduce the *Local Markov Property* which allows one to deduce conditional independency statements from a Bayesian network. We will not go into the proof of the theorem as it goes beyond scope of our study, however a proof may be found in the book by Koski & Noble [19].

Definition 3.3.2. (Local Markov Property)
Let $\mathcal{G} = (V, D)$ be a DAG with vertices $V = \{X_1, ..., X_d\}$. A probability density function P over the random vector $\mathbf{X} = (X_1, ..., X_d)$ satisfies the *Local Markov Property* if for each $j \in \{1, ..., d\}$,

$$X_j \perp V \backslash \{V_j \cup \Pi_j\} | \Pi_j.$$

That is, X_j is conditionally independent of its non-descendants given its parents.

In a BN, the presence of a directed edge from node A to node B indicates that A has an influence over B. It is important to note that A does not necessarily cause B, however the change in certainty of the value of A may change the certainty of the value of B. BNs may be used to determine how evidence on the state of a variable C may influence the belief we have on the state of a variable D, even if C is not a parent of D. We present three basic BN structures and discuss how information flows in these DAGs through the use of examples.

Converging Connections

Suppose an investor wants to decide whether to invest in a particular stock or not. Let's say the said investor is particularly interested in the following set of variables $V = \{X_1, X_2, X_3\}$ where

- X_1 is the binary classification variable telling the investor whether they should invest or not.
- X_2 is a categorical variable which denotes which industry the company is in.
- X_3 is a categorical variable denoting whether the P\E ratio lies in the interval $[0, 10), [10, 20)$ or $[20, \infty)$.
  Note: P\E ratio $= \frac{\text{Share Price}}{\text{Earnings Per Share}}$ where

$$\text{Earnings Per Share} = \frac{\text{Total Company Earnings}}{\text{No of shares the company has issued for sale}}$$

Let's say that the DAG in figure 3.3.1 is obtained after performing structure learning on a provided data set. Such a DAG is an example of a converging connection. From the figure,using the Local Markov Property, we can deduce that $X_2 \perp X_3|\emptyset$, meaning that the industry in which the company operates is independent of its P\E ratio. The DAG also imposes that both X_2 and X_3 influence X_1. Hence the investor should consider both the industry and P\E ratio of the company before making any investment decision. The factorization of the joint probability distribution along the DAG is $P_{X_1,X_2,X_3} = P_{X_1|X_2,X_3}P_{X_2}P_{X_3}$.

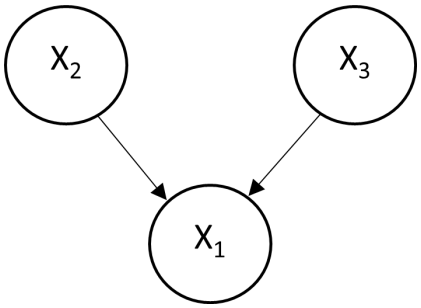

Figure 3.3.1: A DAG on three variables

Note that the DAG in figure 3.3.1 gives us the following parent sets

$$\Pi_1 = \{X_2, X_3\}, \Pi_2 = \emptyset \text{ and } \Pi_3 = \emptyset$$

Serial Connections

Suppose we are interested in three discrete variables pertaining to online gamblers; their country of residence (C), their total deposits (D) and the amount of money (M) the player has lost. An example of a serial connection is given in figure 3.3.2. Here the country of the gambler is believed to have an influence on the depositing behaviour of the gambler which in turn influences the amount of money the gambler loses. Thus, suppose the country of residence of the gambler is known but nothing

is known on the amount of money he has deposited nor the amount of money he has lost. The serial connection allows one to use the information given on C to evaluate the uncertainty of D and in turn use this to evaluate the uncertainty of M. So information given on C led to more knowledge on M when D was unknown. Suppose instead that D is known. Using the Local Markov Property, we can say that $M \perp C|D$. This means that given that we have information regarding the deposits of a player, also knowing the gambler's country does not yield relevant information on the amount of money the player has lost.

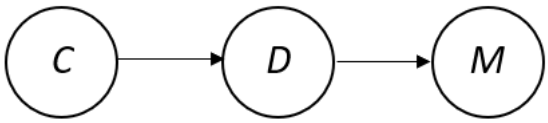

Figure 3.3.2: Serial Connection example

Diverging Connections

For example, suppose there are 2 different symptoms of some virus, labelled S_1, S_2. Let C be a binary variable indicating whether a person has the virus ($C = 1$) or not ($C = 0$) and $X_i = 1$ if an individual has symptom S_i and 0 otherwise. Then it makes sense to consider the DAG in figure 3.3.3 for this scenario because whether or not a person has the virus will affect the probability of said person to have certain symptoms. Such a DAG is an example of a diverging connection. Now, using the Local Markov Property, from the DAG below we can deduce that if we know whether the person has the virus or not, knowing that he has symptom S_1 does not change our belief on the probability that he has symptom S_2, and vice versa.

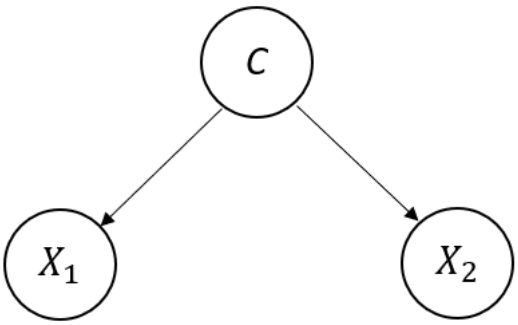

Figure 3.3.3: Diverging Connection example

3.4 Bayesian Network Classifiers

Having defined a Bayesian network and given examples of its basic structures, we now move on to define Bayesian network classifiers. A Bayesian network classifier is a Bayesian network used for predicting a discrete class variable C with range given by $\mathcal{C}$. Given an observation $\boldsymbol{x}$ of a random vector $\boldsymbol{X} = (X_1, ..., X_d)$ the Bayesian classifier aims to find the most probable class c^* given by

$$c^* = \underset{c \in \mathcal{C}}{\operatorname{argmax}} P(c|\boldsymbol{x}) = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \frac{P(\boldsymbol{x}; c)}{P(\boldsymbol{x})} = \underset{c \in \mathcal{C}}{\operatorname{argmax}} P(c; \boldsymbol{x})$$

The classifier factorizes the joint distribution $P(\boldsymbol{x}; c)$ according to a Bayesian network $\mathcal{B} = (\mathcal{G}, P)$. More explicitly,

$$P(\boldsymbol{x}; c) = P(c|\Pi_C) \prod_{i=1}^{d} P(x_i|\Pi_i)$$

where Π_i is as defined in definition 3.3.1 and Π_C is the parent set of C in the DAG $\mathcal{G}$.

The simplest and perhaps most widely used Bayesian network classifier which we shall now introduce is the Naïve Bayes classifier proposed by Minsky in 1961 [20]. More advanced BN classifiers will be discussed in section 3.5.

3.4.1 Naïve Bayes Classifier

Given predictor variables $X_1, ..., X_d$ and a categorical dependent variable C, the Naïve Bayes (NB) model, proposed by Minsky, makes the strong assumption that all predictor variables are conditionally independent given C. In reality, it is very improbable that all predictors will be conditionally independent. The importance of the NB model to the rest of the chapter is due to its use in the FSSJ algorithm explained in section 3.5.2. We will see how the FSSJ algorithm aims to improve upon the NB model by relaxing the conditional independence assumption imposed by the latter model.

The conditional independence assumption asserts that, for all $i = 1, ..., d$,

$$P_{X_i|X_1,...,X_{i-1},X_{i+1},...,X_d,C}(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_d, c) = P_{X_i|C}(x_i|c)$$

Thus, using the conditional independence assumption, the joint probability

$$P_{X_1,...,X_d,C}(x_1, ..., x_d, c)$$

may be written as

$$\begin{aligned}
&P_{X_1,...,X_d,C}(x_1, ..., x_d, c) \\
&= P_{X_1|X_2,...,X_d,C}(x_1|x_2, ..., x_d, c)P_{X_2,...,X_d,C}(x_2, ..., x_d, c) \\
&= P_{X_1|X_2,...,X_d,C}(x_1|x_2, ..., x_d, c)P_{X_2|X_3,...,X_d,C}(x_2|x_3, ..., x_d, c)P_{X_3,...,X_d,C}(x_3, ..., x_d, c) \\
&= ... \\
&= P_{X_1|X_2,...,X_d,C}(x_1|x_2, ..., x_d, c)P_{X_2|X_3,...,X_d,C}(x_2|x_3, ..., x_d, c) \times ... \\
&\times P_{X_{d-1}|X_d,C}(x_{d-1}|x_d, c)P_{X_d|C}(x_d|c)P_C(c) \\
&= P_{X_1|C}(x_1|c)P_{X_2|C}(x_2|c)...P_{X_{d-1}|C}(x_{d-1}|c)P_{X_d|C}(x_d|c)P_C(c)
\end{aligned}$$

Hence the DAG of the Naïve Bayes model is as in figure 3.4.1

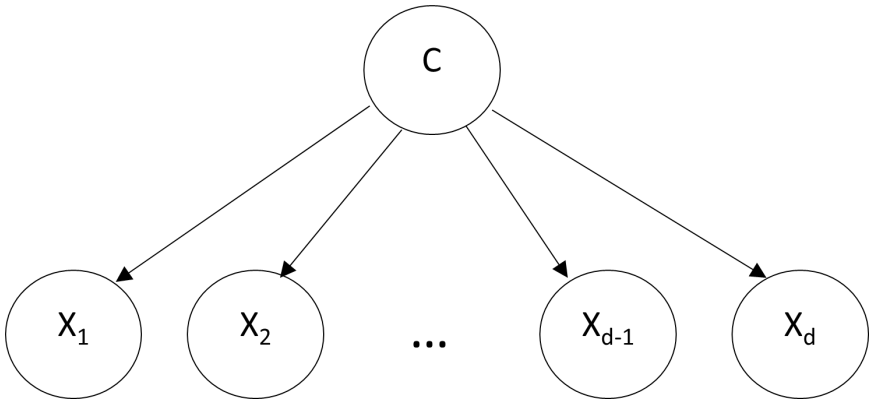

Figure 3.4.1: Naïve Bayes Bayesian network

Classification using the Naïve Bayes classifier is done in the following manner.

Consider $\mathcal{Z}$ as defined in section 3.2. As explained previously, we want to find the class c^* which maximises $P(\boldsymbol{x}; c) = P(\boldsymbol{x}; c) = P(c)\prod_{i=1}^{d} P(x_i|c)$. To estimate

the latter joint distribution we replace the actual probabilities with their empirical estimates (see definitions 3.2.1 and 3.2.2). Hence we assign an observation $\boldsymbol{x}$ to the class label

$$c^* = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \hat{P}(c) \prod_{i=1}^{d} \hat{P}(x_i|c)$$

where $\hat{P}(c)$ is estimated by the number of observations in $\mathcal{Z}$ with the class label equal to c divided by n.

As mentioned earlier, the assumption that all predictor variables are conditionally independent given the class variable is quite unrealistic. Consider, for example, the following fictitious scenario where we want to predict whether a gambler is a problem gambler or not using the predictor variables; income, age and education level. It seems counter-intuitive to ignore the correlations between income and education level and assume that they are conditionally independent given that we know whether the player is a problem gambler or not. Due to such issues stemming from the strong independence assumption of the NB classifier, attempts have been made to improve the performance of Bayesian classifiers by relaxing the strong assumption imposed by the NB approach. We discuss two such attempts, namely the adaptation of the Chow-Liu algorithm by Friedman et al. [24] and the FSSJ algorithm by Pazzani [25], in the next section.

3.5 Structure learning

A structure learning algorithm, in the context of Bayesian networks, takes as input a data set and aims to fit the DAG which best explains the underlying relationships between the variables of interest. In this section we will be discussing two different structure learning techniques which were then implemented on our training data set in chapter 5. These are the Chow-Liu adaptation by Friedman et al. [24] and the FSSJ (Forward Sequential Selection and Joining) algorithm by Pazzani [25].

3.5.1 Chow-Liu Algorithm

In this section we will first discuss the Chow-Liu (CL) algorithm for structure learning of a Bayesian network. We will then present the adaptation of the CL algorithm to Bayesian classifiers by Friedman et al. [24].

The following two definitions will be useful later on in the section, specifically in sub section 3.5.1.1 where we introduce an adaptation of the Chow-Liu algorithm.

Definition 3.5.1. A DAG with nodes representing the discrete random variables $X_1, ..., X_d$ is a *tree* if each X_i has only one parent except for one variable, referred to as the root node, that has no parents.

Note that a tree has no directed cycles. An example of a tree is given in figure 3.5.1. Note that X_4 is the root node.

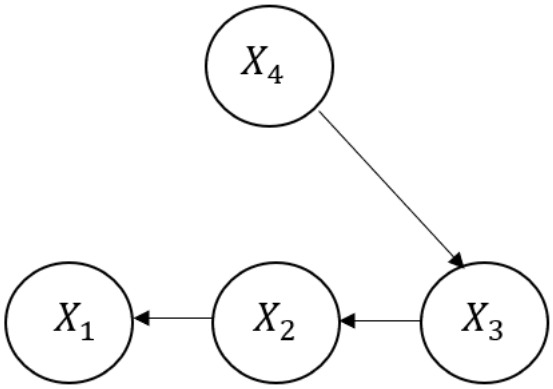

Figure 3.5.1: Example of a tree

Definition 3.5.2. Let $s(j)$ be the indexing set of the parent set Π_j defined in definition 3.2.3 .
Then s is said to *define* a tree over $X_1, ..., X_d$ if there is exactly one i such that $s(i) = 0$ (this i refers to the root node of the tree), and there is no sequence $i_1, ..., i_k$ such that $s(i_j) = i_{j+1}$ for $i \leq j < k$ and $s(i_k) = i_1$ (i.e, there are no directed cycles).

Consider the directed graph given in figure 3.5.2. Let $i_1 = 1, i_2 = 2$ and $i_3 = 3$. Then since X_2 is a parent of X_1, $s(i_1) = i_2$. Similarly, $s(i_2) = i_3$ and $s(i_3) = i_1$.

Thus i_1, i_2, i_3 is a sequence such that $s(i_j) = i_{j+1}$ for $i \leq j < 3$ and $s(i_3) = i_1$. Thus s does not define a tree over $X_1, .., X_4$ in the case of the directed graph in figure 3.5.2.

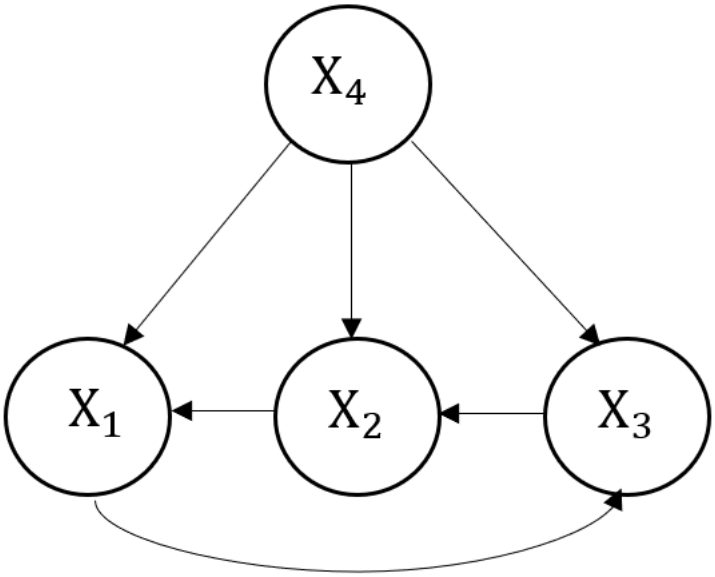

Figure 3.5.2: Example of a directed graph which is not tree

The Chow-Liu algorithm makes the assumption that each variable has at most one parent. Thus, the structure of the DAG returned by the CL algorithm is a tree. In general, such an assumption imposed by the CL algorithm may be unrealistic since there are countless scenarios where a variable is believed to be influenced by two or more other variables . However, as we will see later on in lemma 3.5.2 , such an assumption yields the advantage of the existence of an algorithm which finds the optimal tree. Given d variables the Chow-Liu approach is to find the $d-1$ dependencies which maximize the likelihood $P(\mathcal{D}|\mathcal{G}, \boldsymbol{\psi})$. Let $\Pi_i^{\mathcal{G}}$ denote the parent of X_i under graph structure $\mathcal{G}$. Since we have d variables with $d-1$ dependencies and each variable may be conditioned on at most one other variable, the distribution over the random vector $\mathbf{X} = (X_1, ..., X_d)$ is of the form

$$P_{X_1,...,X_d} = \prod_{i=1}^{d} P_{X_i|\Pi_i^{\mathcal{G}}}$$

In order to explain how the Chow-Liu algorithm works, we will need to define some key notions.

Definition 3.5.3. (Mutual Information) The mutual information between two discrete random vectors X and Y is defined as

$$I(X,Y) = \sum_{x,y} P_{X,Y}(x,y) log \frac{P_{X,Y}(x,y)}{P_X(x)P_Y(y)}$$

.

Note that

$$I(X,Y) = I(Y,X). \tag{3.1}$$

Let $\hat{I}(X,Y) = \sum_{x,y} \hat{P}_{X,Y}(x,y) log \frac{\hat{P}_{X,Y}(x,y)}{\hat{P}_X(x)\hat{P}_Y(y)}$ denote the empirical estimate of the mutual information where the actual pdfs are replaced by their empirical estimates. Note that the definition holds if the variables in consideration are multivariate.

Definition 3.5.4. (Maximum Weight Dependence Tree) A tree σ is said to be a maximum weight dependence tree if for any other tree $\hat{\sigma}$,

$$\sum_{i=1}^{d} \hat{I}(X_i, \Pi_i^{\sigma}) \geq \sum_{i=1}^{d} \hat{I}(X_i, \Pi_i^{\tilde{\sigma}})$$

where $(X_i, \Pi_i^{\sigma})_{i=1}^{d}$ denotes the edge set of σ and $(X_i, \Pi_i^{\tilde{\sigma}})$ of $\tilde{\sigma}$.

We now introduce Kruskal's algorithm for finding a Maximum Weight Dependence Tree. This algorithm is critical to guarantee the optimality of the CL algorithm mentioned in lemma 3.5.2.

Algorithm 1: Kruskal's Algorithm

1. Calculate the mutual information for each of the $\frac{d(d-1)}{2}$ edges and index them in decreasing order with respect to their weights (mutual information) $b_1, b_2, ..., b_{\frac{d(d-1)}{2}}$.
2. Select edges b_1 and b_2 and add edge b_3 if it does not form a cycle. Discard edge b_3 if it forms a cycle.
3. Repeat this for edges $b_4, b_5, ..., b_{\frac{d(d-1)}{2}}$ in that order, adding edges if they do not form a cycle and discarding them if they do.

Lemma 3.5.1. Kruskal's algorithm returns the tree with maximum weight.

Proof. The lemma is proved by induction on the number of nodes of vertices of the graph.

The result is clearly true for two nodes.

Now, assume that the result holds for d nodes and consider a collection of $d+1$ nodes labelled $(X_1, ..., X_{d+1})$ where they are ordered such that for each $j = 1, ..., d+1$, the maximal tree using variables $(X_1, ..., X_j)$ gives the maximal tree from any selection of j nodes from the full set of $d+1$ nodes. So if we pick say $j = 6$ and say $d = 10$, then the maximal tree using variables $(X_1, ..., X_6)$ has larger weight than any other maximal tree composed using 6 of the 10 available variables.

Let $b_{(i,j)}$ denote the weight of the edge (i, j) and consider the edges to be undirected.

Let $\mathcal{T}_j^{(d+1)}$ denote the maximal tree obtained by selecting j nodes from the $d+1$ available. Consider $\mathcal{T}_{d+1}^{(d+1)}$.

Let Z denote the leaf node in $\mathcal{T}_{d+1}^{(d+1)}$ such that among all leaf nodes in $\mathcal{T}_{d+1}^{(d+1)}$, the edge (Z, Y) in $\mathcal{T}_{d+1}^{(d+1)}$ has the smallest weight.

Removing the node Z gives the maximal tree on d nodes from the set of $d+1$ nodes. This is seen as follows. Clearly, there is no tree with larger weight that can be

formed with these d nodes, otherwise the tree on d nodes with larger weight, with the addition of the leaf (Z, Y) would be a tree on $d+1$ nodes with greater weight than $\mathcal{T}_{d+1}^{(d+1)}$. It follows that the maximal tree on d nodes is formed by $\{X_1, ..., X_d\}$ (due to the way the nodes are originally labelled) and hence $Z = X_{d+1}$. This means that X_{d+1} is a leaf node of $\mathcal{T}_{d+1}^{(d+1)}$. We now show that we can obtain this tree using Kruskal's algorithm.

By the inductive hypothesis, $\mathcal{T}_d^{(d+1)}$ may be obtained by applying Kruskal's algorithm to the weights $(b_{(i,j)})_{1\leq i<j\leq d}$.
Now consider an application of Kruskal's algorithm to the weights $(b_{(i,j)})_{1\leq i<j\leq d+1}$ and note that for any (i, j) with $i < j$ such that the undirected edge (X_i, X_j) forms part of the tree $\mathcal{T}_d^{(d+1)}$, $(b_{(i,d+1)}) < (b_{(i,j)})$ and $(b_{(j,d+1)}) < (b_{(i,j)})$ by the definition of $\mathcal{T}_d^{(d+1)}$.

Therefore if the edges $(b_{(i,j)})_{1\leq i<j\leq d+1}$ are listed according to their weight and the Kruskal algorithm is applied, then all the edges used in $\mathcal{T}_d^{(d+1)}$ will be included in the algorithm before the edges $(b_{(k,d+1)})_{k=1}^{d}$ are considered.

It follows that $\mathcal{T}_{d+1}^{(d+1)}$ is the graph obtained by applying Kruskal's algorithm to the nodes $(X_1, ..., X_{d+1})$.

□

Therefore, if we give each edge (i, j) a weight equal to $I(X_i, X_j)$, i.e we set $b_{(i,j)} = I(X_i, X_j)$ in lemma 3.5.1 above, Kruskal's algorithm certifies that we will be able to find a maximum weight dependence tree. Having defined the necessary notions and presented Kruskal's algorithm, we are now in a position to see how the CL algorithm is implemented. We then demonstrate how the necessary steps are carried out through the use of an example.
Note that if all weights are different, this procedure returns a unique tree with $d-1$ edges. If two weights are equal one may impose an arbitrary ordering. We now have a look at an example of the CL algorithm

Algorithm 2: Chow-Liu Algorithm

1. Compute $\hat{I}(X_i, X_j)$ for each pair of variables X_i, X_j where $i \neq j$.
2. Build a complete undirected graph where the nodes represent the variables $X_1, ..., X_d$. Label the weight of an edge connecting X_i to X_j by $\hat{I}(X_i, X_j)$.
3. Find the maximum weighted spanning tree using Kruskal's algorithm.
4. Transform the resulting undirected tree to a directed one by choosing a root node and setting the direction of all edges to be outward from it.

Example 3.5.1. (Chow-Liu Algorithm).

In this example we shall first implement Kruskal's algorithm to find the maximum weight spanning tree, and then complete the Chow-Liu algorithm by transforming the undirected edges into directed ones so as to obtain the final DAG. Assume we are interested in variables $X_1, \ldots, X_4$. For demonstration purposes, assume that the empirical estimates of the pairwise mutual information values in table 3.1 were obtained from a sample of n observations $\{\boldsymbol{x}_{(1)}, ..., \boldsymbol{x}_{(n)}\}$ of the random vector $\boldsymbol{X} = (X_1, ..., X_4)$. Note that we are uninterested in the values $(\hat{I}(X_i, X_i))_{i=1}^4$ and hence these values are left blank in the table.

$\hat{I}(X_i, X_j)$	X_1	X_2	X_3	X_4
X_1		0.1	0.2	0.2
X_2	0.1		0.4	0.3
X_3	0.2	0.4		0.5
X_4	0.2	0.3	0.5	

Table 3.1: Empirical Estimates of Mutual Information

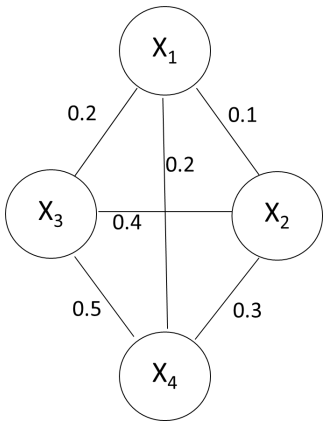

Figure 3.5.3: Assigning Mutual Information Value to each pair of variables

Steps i)-iii) in figure 3.5.4 show how Kruskal's algorithm selects the edges that maximize the weight of the tree. Note that a chosen edge will be coloured in red.

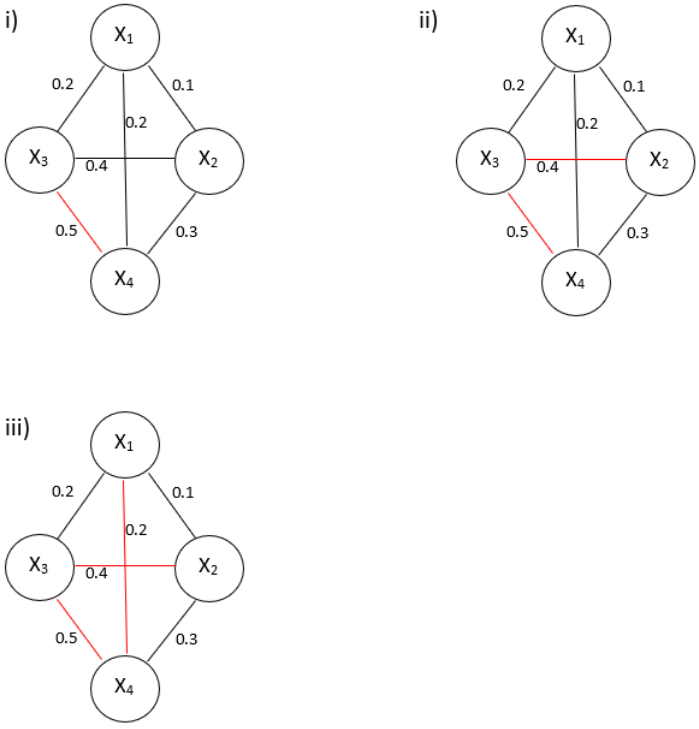

Figure 3.5.4: Selecting edges which maximize the weight of the tree

To complete the Chow-Liu algorithm, we now pick a root node and add directed edges to the tree. If we pick X_3 as the root node, this gives us the DAG in figure 3.5.5.

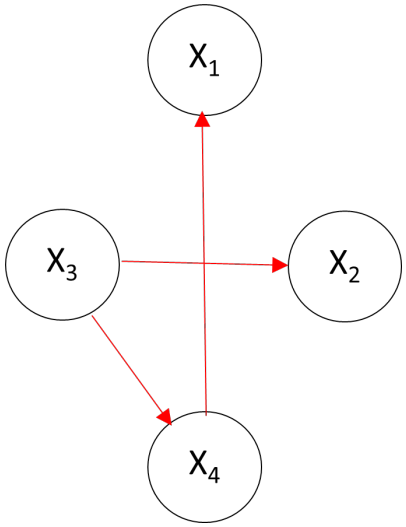

Figure 3.5.5: Final DAG Obtained by Chow-Liu algorithm

We shall now start working towards proving one of the main results of this section, namely theorem 3.5.2. We first attempt to ease the notation which will be used throughout the rest of the section. Recall from definition 3.2.3 that $s(j)$ denotes the index of the parent set of X_j. In the Chow-Liue algorithm, each variable is assumed to have at most one parent, hence $\Pi_j = X_{s(j)}$ if X_j has a parent and $\Pi_j = \emptyset$ otherwise.

First, set

$$n_k(x_j^{(i)}, x_{s(j)}^{(l)}) = \begin{cases} 1 & \text{if } (x_j^{(i)}, x_{s(j)}^{(l)}) \text{ is found in } \mathbf{x}_{(k)} \\ 0 & \text{otherwise} \end{cases}$$

where $(x_j^{(i)}, x_{s(j)}^{(l)})$ is an observation of the variables (X_j, Π_j).

Let $\boldsymbol{\theta}$ denote the set of conditional probabilities, which we will refer to as parameters and which are defined as follows

$$\theta_{jil} = P(\{X_j = x_j^{(i)}\} | \{\Pi_j = x_{s(j)}^{(l)}\}) \tag{3.2}$$

for $j = 1, ..., d, i = 1, ..., k_j, l = 1, ..., k_{s(j)}$ and for a given graph structure $\mathcal{G} = (V, D)$.

θ_{jil} may be interpreted as the probability that X_j takes the value $x_j^{(i)}$ given that its parents' configuration is l. These parameters may be referred to as the conditional probability distributions (CPDs). We now derive the likelihood function of $\mathcal{D}$ given a graph structure $\mathcal{G}$.

The probability of observing $\mathbf{x}_{(k)}$ may be written as

$$\begin{aligned} P_{X|\mathbf{\Psi}}(\mathbf{x}_{(k)}|\boldsymbol{\psi},\mathcal{G}) &= \prod_{j=1}^{d}\prod_{l=1}^{k_{s(j)}}\prod_{i=1}^{k_j} P(\{X_j = x_j^{(i)}\}|\{\Pi_j = x_{s(j)}^{(l)}\})^{n_k(x_j^{(i)},x_{s(j)}^{(l)})} \\ &= \prod_{j=1}^{d}\prod_{l=1}^{k_{s(j)}}\prod_{i=1}^{k_j} \theta_{jil}^{n_k(x_j^{(i)},x_{s(j)}^{(l)})} \end{aligned}$$

Since the observations $\mathbf{x}_{(1)},...,\mathbf{x}_{(\mathbf{n})}$ forming $\mathcal{D}$ are considered to be independent observations, we get that

$$\begin{aligned} P_{X|\mathbf{\Psi}}(\mathcal{D}|\boldsymbol{\psi},\mathcal{G}) &= \prod_{k=1}^{n} P_{X|\mathbf{\Psi}}(\mathbf{x}_{(k)}|\boldsymbol{\psi},\mathcal{G}) \\ &= \prod_{j=1}^{d}\prod_{l=1}^{k_{s(j)}}\prod_{i=1}^{k_j}\prod_{k=1}^{n} \theta_{jil}^{n_k(x_j^{(i)},x_{s(j)}^{(l)})} \end{aligned}$$

Set $n(x_j^{(i)},x_{s(j)}^{(l)}) = \sum_{k=1}^{n} n_k(x_j^{(i)},x_{s(j)}^{(l)})$ which counts the number of times the configuration $(x_j^{(i)},x_{s(j)}^{(l)})$ appears in $\mathcal{D}$.

The likelihood may therefore be written as

$$L(\mathcal{D}|\boldsymbol{\psi}) = \prod_{i=1}^{n} P_{X|\mathbf{\Psi}}(\mathbf{x}_{(k)}|\boldsymbol{\psi},\mathcal{G}) = \prod_{j=1}^{d}\prod_{l=1}^{k_{s(j)}}\prod_{i=1}^{k_j} \theta_{jil}^{n(x_j^{(i)},x_{s(j)}^{(l)})}$$

Theorem 3.5.2. *Let* $\mathcal{D} = \left(\boldsymbol{x}_{(1)}^T,...,\boldsymbol{x}_{(n)}^T\right)$. *Then the CL algorithm maximises the likelihood* $L(\mathcal{D}|\boldsymbol{\psi})$.

Proof. The log-likelihood may be written as

$$
\begin{aligned}
l(\mathcal{D}|\boldsymbol{\psi}) &= \sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \log \theta_{jil}^{n(x_j^{(i)}, x_{s(j)}^{(l)})} \\
&= \sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} n(x_j^{(i)}, x_{s(j)}^{(l)}) \log \theta_{jil} \\
&= n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \frac{n(x_j^{(i)}, x_{s(j)}^{(l)})}{n} \log P(X_j = x_j^{(i)} | \Pi_j = x_{s(j)}^{(l)}) \\
&= n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \left(\frac{\hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)})}{\hat{P}(X_{s(j)} = x_{s(j)}^{(l)})} \frac{\hat{P}(X_j = x_j^{(i)})}{\hat{P}(X_j = x_j^{(i)})} \right) \\
&= n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \frac{\hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)})}{\hat{P}(X_{s(j)} = x_{s(j)}^{(l)})\hat{P}(X_j = x_j^{(i)})} \hat{P}(X_j = x_j^{(i)}) \\
&= n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \frac{\hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)})}{\hat{P}(X_{s(j)} = x_{s(j)}^{(l)})\hat{P}(X_j = x_j^{(i)})} \\
&+ n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \hat{P}(X_j = x_j^{(i)}) \qquad (3.3)
\end{aligned}
$$

Now, since

$$
\begin{aligned}
& n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \hat{P}(X_j = x_j^{(i)}) \\
&= n\sum_{j=1}^{d}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}) \log \hat{P}(X_j = x_j^{(i)})
\end{aligned}
$$

does not depend on the graph structure, to maximize the log-likelihood in 3.3 we need to maximize

$$
n\sum_{j=1}^{d}\sum_{l=1}^{k_{s(j)}}\sum_{i=1}^{k_j} \hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)}) \log \frac{\hat{P}(X_j = x_j^{(i)}, X_{s(j)} = x_{s(j)}^{(l)})}{\hat{P}(X_{s(j)} = x_{s(j)}^{(l)})\hat{P}(X_j = x_j^{(i)})} = n\sum_{j=1}^{d} \hat{I}(X_j, X_{s(j)})
$$

Hence, since the Chow-Liu algorithm finds the graph structure which maximizes the mutual information

$$
\sum_{j=1}^{d} \hat{I}(X_j, X_{s(j)})
$$

it follows that it also maximizes the likelihood.

□

Remark. Recall from definition 3.5.3 that $I(X, Y) = I(Y, X)$. Hence, the edge directions do not matter for the likelihood.

3.5.1.1 Adaptation of the Chow-Liu algorithm

In this subsection we discuss how to adapt the CL algorithm to return a Bayesian classifier which still maximises the likelihood. We now assume that the data set in consideration is $\mathcal{Z}$.

Definition 3.5.5. (Tree Augmented Naïve Bayesian network) We say that a Bayesian network, over predictor variables $X_1, ..., X_d$ and class variable C with DAG G has a Tree Augmented Naïve (TAN) structure if C has no parents and each attribute has as parents C and at most one other attribute. More rigorously, $\Pi_C = \emptyset$, $\Pi_i = \{C, X_{(s(i))}\}$ if $s(i) > 0$ and $\Pi_i = C$ if $s(i) = 0$.

As mentioned earlier in section 3.4.1, the Naïve Bayes independence assumption is quite unrealistic in general. To improve the classifier, Friedman et al. [24] proposed augmenting the Naïve Bayesian network by adding edges between the predictor variables thus introducing conditional dependencies between the attributes. The main issue with this approach is that adding the best set of augmenting edges is an intractable problem since it is equivalent to learning the best Bayesian network among which C is a root node, which is an NP-hard problem as proved by Chickering [10]. Therefore, they impose the assumption that each predictor variable will have as parents the class variable C, as in the case of the Naïve Bayesian network, and at most one other parent. This implies that the structure of the Bayesian network returned by this algorithm will be TAN. From now on, we will refer to the adaptation of the CL algorithm as CL-adapt. It turns out, as proved by Friedman et al. [24], that despite such a strong assumption, the network returned by the adaptation of the

CL algorithm still maximises the likelihood. Before we can delve into the algorithm, we need to familiarise ourself with the following definition.

Definition 3.5.6. (Conditional Mutual Information) The conditional mutual information between variables X and Y given the variable Z is defined as

$$I(X, Y|Z) = \sum_{x,y,z} P(x, y, z) \frac{P(x, y|z)}{P(x|z)P(y|z)}$$

We will denote the empirical estimate of $I(X, Y|Z)$ as $\hat{I}(X, Y|Z)$.The procedure of the CL-adapt algorithm follows closely the general outline of the CL algorithm. However, since each predictor variable will have the class variable as a parent, instead of considering the mutual information between attributes, we will take into account their conditional information given the class variable C. The following is an outline of the steps involved in the CL-adapt algorithm.

Algorithm 3: CL-adapt algorithm

1. Compute $\hat{I}(X_i, X_j|C)$ for each pair of attributes X_i, X_j where $i \neq j$.
2. Build a complete undirected graph where the nodes represent the variables $X_1, ..., X_d$. Annotate the weight of an edge connecting X_i to X_j by $\hat{I}(X_i, X_j|C)$.
3. Find the maximum weighted spanning tree using Kruskal's algorithm.
4. Transform the resulting undirected tree to a directed one by choosing a root node and setting the direction of all the edges to be outward from it.
5. Construct a TAN model by adding a vertex labelled C and adding a directed edge from C to each predictor variable.

Before we can move on to the next theorem, some definitions and results, adapted from the work of Cover et al. [11] are in order. The importance of these notions lie in their use in the proof of theorem 3.5.7.

Definition 3.5.7. (Entropy)
Let $P(X)$ be a distribution over a random variable X. The entropy of X is defined as

$$\mathbf{H}_P(x) = \mathbb{E}_P\left[\log \frac{1}{P(X)}\right] = \sum_{x \in \mathcal{X}} P(x) \log \frac{1}{P(x)}$$

The entropy of a random variable is a measure of the uncertainty associated with that variable. For example, let X be a Bernouilli distributed random variable which models the outcome of a biased coin toss where the probability of tails is equal to 1. The entropy of X in this case would be 0 which reflects that there is no surprise in the outcome of X. The notion of entropy can be generalised to random vectors as we now show.

Definition 3.5.8. Suppose we have a joint distribution over random variables $X_1, ..., X_d$ with corresponding PDF P. Then the joint entropy of $X_1, ..., X_d$ is

$$\mathbf{H}_P(X_1, ..., X_d) = \mathbb{E}_P\left[\log \frac{1}{P(X_1, ..., X_d)}\right]$$

For the sake of convenience, we usually refer to $\mathbf{H}_P(X)$ by $\mathbf{H}(X)$ unless it is unclear over which probability distribution we are taking the expectation.

Definition 3.5.9. (Conditional Entropy) Suppose we have a joint distribution $P_{X,Y}$ over random variables X and Y. The conditional entropy $H(Y|X)$ is defined as

$$\mathbf{H}(Y|X) = \mathbb{E}_{P_{X,Y}}\left[\log \frac{1}{P_{Y|X}}\right]$$

Lemma 3.5.3. With X and Y defined as in definition 3.5.9, $\mathbf{H}(X,Y) = \mathbf{H}(X) + \mathbf{H}(Y|X)$.

Proof. Since $\log P(X,Y) = \log P(Y|X)P(X) = \log P(Y|X) + \log P(X)$,we get that

$$\begin{aligned}
\mathbf{H}(X,Y) &= \mathbb{E}_{PX,Y}\left[\log \frac{1}{P(X,Y)}\right] \\
&= -\mathbb{E}_{PX,Y}[\log P(X,Y)] \\
&= -\left(\mathbb{E}_{P_{X,Y}}[\log P(Y|X) + \log P(X)]\right) \\
&= -\left(\mathbb{E}_{P_{X,Y}}[\log P(Y|X)] + \mathbb{E}_{P_{X,Y}}[\log P(X)]\right) \\
&= \mathbf{H}(X) + \mathbf{H}(Y|X)
\end{aligned}$$

□

Using similar steps as those outlined in the lemma directly above, the following corollary can be easily proven to be true.

Corollary 3.5.4. $\mathbf{H}(X,Y|Z) = \mathbf{H}(X|Z) + \mathbf{H}(Y|X,Z)$.

We may rewrite the definition of $I(X,Y)$ as follows.

$$\begin{aligned}
I(X,Y) &= \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \\
&= \sum_{x,y} P(x,y) \log \frac{P(x|y)}{P(x)} \\
&= -\sum_{x,y} P(x,y) \log P(x) - \left(-\sum_{x,y} P(x,y) \log P(x|y)\right) \\
&= \mathbf{H}(X) - \mathbf{H}(X|Y) \\
&= \mathbf{H}(Y) - \mathbf{H}(Y|X) \text{ (by 3.1)}
\end{aligned}$$

Note that combining the result above with lemma 3.5.3 we get that $I(X,Y) = \mathbf{H}(X) + \mathbf{H}(Y) - \mathbf{H}(X,Y)$. In particular, $I(X,X) = \mathbf{H}(X)$. This definition of mutual information makes the proof of the following theorem very simple.

Theorem 3.5.5. *(Chain rule for entropy) Suppose we have a joint distribution over random variables $X_1, ..., X_d$ with corresponding PDF P. Then*

$$\mathbf{H}(X_1, ..., X_d) = \sum_{i=1}^{d} \mathbf{H}(X_i | X_{i-1}, ..., X_1)$$

Proof. By repeated use of lemma 3.5.3, we have that

$$\begin{aligned}
\mathbf{H}(X_1, X_2) &= \mathbf{H}(X_1) + \mathbf{H}(X_2|X_1) \\
\mathbf{H}(X_1, X_2, X_3) &= \mathbf{H}(X_1) + \mathbf{H}(X_2, X_3|X_1) = \mathbf{H}(X_1) + \mathbf{H}(X_2|X_1) + \mathbf{H}(X_3|X_2, X_1) \\
\mathbf{H}(X_1, ..., X_d) &= \mathbf{H}(X_1) + \mathbf{H}(X_2|X_1) + \ldots + \mathbf{H}(X_d|X_{d-1}, ..., X_1) \\
&= \sum_{i=1}^{d} \mathbf{H}(X_i|X_{i-1}, ..., X_1)
\end{aligned}$$

□

The following lemma is critical in the proof of theorem 3.5.7.

Lemma 3.5.6. (Chain rule for information) Given random variables $X_1, ..., X_d, Y$ we have that

$$I(X_1, ..., X_d; Y) = \sum_{i=1}^{d} I(X_i; Y | X_{i-1}, X_{i-2}, ..., X_1)$$

Proof.

$$\begin{aligned}
&I\left(X_1, X_2, \ldots, X_d; Y\right) \\
&\quad = H\left(X_1, X_2, \ldots, X_d\right) - H\left(X_1, X_2, \ldots, X_d \mid Y\right) \\
&\quad = \sum_{i=1}^{d} H\left(X_i \mid X_{i-1}, \ldots, X_1\right) - \sum_{i=1}^{d} H\left(X_i \mid X_{i-1}, \ldots, X_1, Y\right) \\
&\quad = \sum_{i=1}^{d} I\left(X_i; Y \mid X_1, X_2, \ldots, X_{i-1}\right).
\end{aligned}$$

□

With the chain rule for information proven, we are now in a position to prove the most important result of this sub section.

Theorem 3.5.7. *Let $\mathcal{Z} = \left((\boldsymbol{x}_{(1)}, c_{(1)}), ..., (\boldsymbol{x}_{(n)}, c_{(n)})\right)$ be a data set containing n independent observations. The procedure CL-adapt maximises the likelihood $L(\mathcal{Z}|\boldsymbol{\psi})$.*

Proof. We showed in theorem 3.5.2 that given a data set containing n independent observations the log-likelihood is given by

$$L(\mathcal{D}|\boldsymbol{\psi}) = n \sum_{j=1}^{d} \hat{I}(X_j, X_{s(j)}) + K$$

where K is a constant term which does not depend on the graph structure. Generalizing this result for $\mathcal{Z}$ as defined in the theorem, we get that

$$L(\mathcal{Z}|\boldsymbol{\psi}) = n \left(\sum_{j=1}^{d} \hat{I}(X_j, X_{s(j)}) + \hat{I}(C, \Pi_C) \right) + K'$$

where K' is again a constant that does not depend on the graph structure. Since $\Pi_C = \emptyset, \hat{I}(C, \Pi_C)) = 0$. Since in a TAN model the parents of X_i are specified by the function s, we have that

$$\hat{I}(X_i, \Pi_i) = \begin{cases} \hat{I}\left(X_i, (X_{s(i)}, C)\right), & \text{if } s(i) > 0 \\ \hat{I}(X_i, C), & \text{if } s(i) = 0 \end{cases}$$

Hence, in order to maximise the likelihood, we need to maximise the term

$$\sum_{i, s(i)>0} \hat{I}\left(X_i, (X_{s(i)}, C)\right) + \sum_{i, s(i)=0} \hat{I}(X_i, C) \tag{3.4}$$

By using the chain rule for information (lemma 3.5.6), we get that

$$\sum_{i, s(i)>0} \hat{I}\left(X_i, (X_{s(i)}, C)\right) = \sum_{i, s(i)>0} \hat{I}(X_i, C) + \hat{I}(X_i, X_{s(i)}|C)$$

and hence we can rewrite expression 3.4 as

$$\sum_{i} \hat{I}\left(X_i, C\right) + \sum_{i, s(i)>0} \hat{I}(X_i; X_{s(i)}|C)$$

Now note that the first term is not affected by the choice of the tree defining function s. Hence, it suffices to maximise the second term. However, since the CL-adapt algorithm finds the graph structure which maximizes the conditional mutual information $\sum_{i,s(i)>0} \hat{I}(X_i; X_{s(i)}|C))$, it follows that it also maximizes the likelihood. □

3.5.2 FSSJ algorithm

In this section we will explain how the FSSJ (Forward Sequential Selection and Joining) algorithm is used to find the structure of a Bayesian network given data D where the dependent variable is the class C. The FSSJ algorithm aims to improve upon the Naïve Bayes classifier by relaxing the strong independence assumption. The FSSJ algorithm, unlike the CL-adapt algorithm, does not set a restriction on the maximum number of parents an attribute can have. Similar to the CL-adapt algorithm it requires that the class C is a parent of each predictor variable. This means that the set of possible Bayesian networks that can be returned by the FSSJ algorithm consists of all those networks in which C is a root node, and we mentioned before that finding the optimal such network is an NP hard problem. The FSSJ algorithm is thus, as we will see, a heuristic algorithm. In particular, it is an example of a hill climbing algorithm, meaning that it starts off with an initial solution and iteratively aims to improve upon the solution by making a change at each step. It is also a greedy algorithm since once a change is made, it cannot be undone at a later stage of the algorithm. This algorithm aims to maximize the k-fold cross-validated estimate of the accuracy which we shall explain below before delving into the steps involved in the algorithm. Note that in our implementation of the algorithm, $\mathcal{Z}$ would be the training data set.

3.5.2.1 *k*-fold Cross-Validation

A common approach when fitting statistical learning models to a data set is to divide it into a training data set and a testing data set. The model is then fitted on the training data set but evaluated on the test data set. Because observations in the test data were not used in the fitting of the classifier, the model is less prone to overfitting. However, since this technique requires a large part of the sample to be left out when fitting the model, it is not convenient when the sample size is small. An approach which addresses this problem is that of cross-validation, which is described in algorithm 4 below.

Algorithm 4: k-fold Cross-Validation

Given: a data set $\mathcal{Z}$ and a value k where $1 < k < n$.

Partition the data set $\mathcal{Z}$ into k partitions $\mathcal{Z}_1, ..., \mathcal{Z}_k$

for *i=1:k* **do**

Remove $\mathcal{Z}_i$ from the sample and train the model on the remaining $k-1$ partitions.

Fit the model on $\mathcal{Z}_i$ and obtain the predicted class labels for the observations $\mathbf{x} \in \mathcal{Z}_i$.

end

After the model has been fitted on each partition, each observation will have a predicted class label and hence the cross-validated accuracy is simply the proportion of such labels which are correct.

In the following section we will understand the relevance of the k-fold Cross-validation to the FSSJ algorithm.

3.5.2.2 Implementation of the FSSJ algorithm

The FSSJ algorithm starts off with no attributes (the empty set) and then performs several operations until there is no longer any improvement in cross validated prediction accuracy. It relaxes the NB assumption that all predictor variables are conditionally independent given the class variable C and instead attempts to search for dependencies among pairs of attributes by joining them in the DAG. Note that the term "joining" refers to adding a directed edge between two vertices.

The following is a summary of the FSSJ algorithm. Note that if $\mathcal{G}$ is a graph, then $V(\mathcal{G})$ and $E(\mathcal{G})$ refer to the vertex set and edge set of $\mathcal{G}$ respectively. Also, a network B will be considered an improvement over a network $\tilde{B}$ if the cross-validated accuracy of B is greater than that of $\tilde{B}$ by at least ϵ which is pre-established. Finally, $\mathcal{Z}$ mentioned in the algorithm below would be the training data set.

Algorithm 5: FSSJ algorithm

Given: a data set $\mathcal{Z}$, a value k corresponding to the k-fold cross-validation required to compute the accuracy and a value ϵ which corresponds to the benchmark value for which an increase in accuracy is considered "significant".

Step 1: Start with the subset of attributes being the empty set. At this stage, the classifier predicts the class with the largest observed frequency in the data. Denote this network by B_0. Set $B_{current} = B_0$.

Step 2: For each attribute not in the current subset of attributes, consider adding the attribute to $B_{current}$. Then measure the accuracy of the model when this attribute is added and store the value. Add to the network the attribute A which most improved the accuracy. Set $V(B_{current}) = V(B_{current}) \cup A$ and $E(B_{current}) = E(B_{current}) \cup (C, A)$.

Step 3: Consider joining each attribute not currently used in $B_{current}$ with each attribute currently used, evaluating the k-fold cross-validated predictive accuracy each time two attributes are joined. If joining two attributes P and Q leads to an improvement in accuracy (by at least ϵ), then set $E(B_{current}) = E(B_{current}) \cup (P, Q)$.

Step 4: If the network which emerges after performing steps 2 and 3 (let's call it B_{new}) is an improvement of the previous one, then set $B_{current} = B_{new}$ and go back to step 2. Else, return $B_{current}$ and halt the algorithm.

Thus the main differences between the DAG obtained by the full NB model and that of the FSSJ algorithm are:

- The FSSJ algorithm may yield a DAG which does not make use of all the available predictor variables.
- The FSSJ algorithm may introduce directed edges between predictor variables.

As we stated above, the FSSJ algorithm iteratively adds variables and dependencies

until there is no longer a significant improvement in accuracy. This means that it can be seen as a feature selection algorithm since it selects a subset of variables (and their corresponding dependencies) whose predictive power is not significantly different from a model containing more features.

Example 3.5.2. Example of FSSJ algorithm.

Suppose we have a data set $\mathcal{H}$ with 3 predictor variables called P, Q, R and 1 independent variable called C.

At first, the initial network B_0 consists solely of C. We then add each of the attributes one by one to the network and compute the accuracy. Let's say that out of all the models with only 1 predictor, the one in figure 3.5.6 proved to be the best in terms of cross validated accuracy.

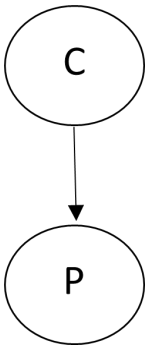

Figure 3.5.6: Optimal network using one predictor

The next step is to separately join Q and R to $P4$ (and obviously also add a directed edge from C into them) and calculate the accuracy to check whether there is any significant improvement. Let's say that the network in figure 3.5.7 is an improvement in terms of accuracy than the one in figure 3.5.6.

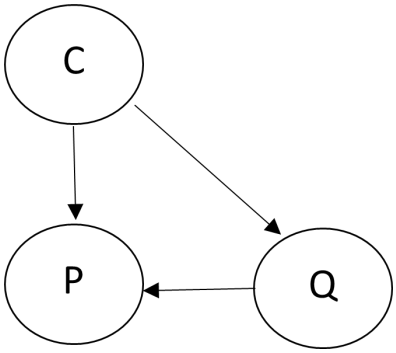

Figure 3.5.7: Optimal network using two predictors

Finally, it remains to check whether R should be included in the network, and if it is, whether it should also have a directed edge pointing to P or $Q4$. We first add R and a directed edge from C to R in the network. Let's say that adding R to the network improves the accuracy but adding directed edges from R to P or Q does not. Then the final network strucure obtained by the FSSJ algorithm can be seen in figure 3.5.8.

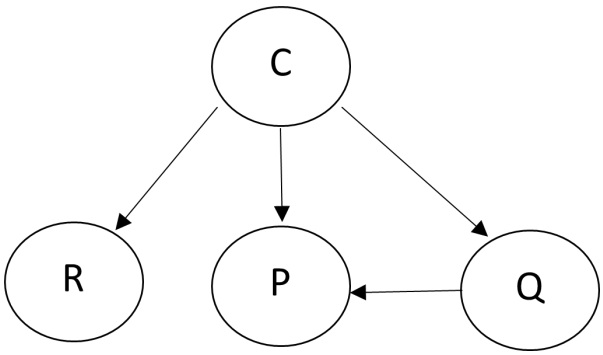

Figure 3.5.8: Final network obtained by FSSJ algorithm

3.6 Parameter Estimation

Given that we know how to obtain the structure of a BN, we now move on to discussing how the conditional probabilities, which we shall refer to as parameters, are estimated from the data given a particular BN. We will be delving into maximum

likelihood estimation as well as Bayesian estimation. We will then derive properties of these estimators in section 3.6.2.1.

To prepare the way for the general BN, we first illustrate how MLE is carried out for Multinomial sampling.

3.6.1 Multinomial Sampling

Suppose that $X_1, ..., X_n \overset{iid}{\sim} X$ where X takes values in a set $\mathcal{X} = \{x^{(1)}, ..., x^{(k)}\}$.
Let $\theta_i = P_X(x^{(i)})$. Hence $\theta_1 + ... + \theta_k = 1$. Set $\boldsymbol{\theta} = (\theta_1, ..., \theta_k)$.
Moreover, let $\mathbf{x} = (x^{(i_1)}, ..., x^{(i_n)})$ be the sequence drawn in n independent trials and let n_l = number of times $x^{(l)}$ appears in $\mathbf{x}$, $l = 1, ..., k$, so that $n = n_1 + ...+, n_k$.

Lemma 3.6.1. The maximum likelihood estimator of θ_j, denoted by $\hat{\theta}_{j,MLE}$ is given by

$$\hat{\theta}_{j,MLE} = \frac{n_j}{n}$$

Proof. $P(\mathbf{x}; \boldsymbol{\theta}) = \theta_1^{n_1}\theta_2^{n_2}...\theta_k^{n_k} = \prod_{i=1}^{k} \theta_i^{n_i}$.

The optimization problem may be written formally as

$$\max_{\boldsymbol{\theta}} \prod_{i=1}^{k} \theta_i^{n_i}$$
$$\text{s.t} \sum_{i=1}^{k} \theta_i = 1 \text{ and}$$
$$\theta_i >= 0 \text{ for } i = 1, ..., k$$

Using the log-likelihood with Lagrange multiplier, we get the function

$\bar{l} = \sum_{i=1}^{k} n_i \log \theta_i + \lambda(1 - \sum_{i=1}^{k} \theta_i)$

Differentiating this and setting it equal to 0 we get

$$\frac{\partial \bar{l}}{\partial \theta_j} = \frac{n_j}{\theta_j} - \lambda = 0 \implies n_j = \lambda\theta_j$$

$$\therefore n = \sum_{j=1}^{k} n_j = \sum_{j=1}^{k} \lambda\theta_j = \lambda$$

Hence we obtain the MLE

$$\hat{\theta}_{j,MLE} = \frac{n_j}{n}$$

□

Remark: $\hat{\theta}_{j,MLE}$ may be easily interpreted as the fraction of times $x^{(j)}$ appears in the data.

3.6.2 Maximum Likelihood Parameter Estimation in a general DAG

Let $V = \{X_1, ..., X_d\}$ and $\mathcal{G} = (V, D)$ denote the DAG along which the probability distribution $P_{X_1,...,X_d}$ is factorized.

Lemma 3.6.2. Given the DAG $\mathcal{G}$ and data set $\mathcal{D}$, the maximum likelihood estimator of θ_{jil}, denoted by $\hat{\theta}_{jil}$ is given by

$$\hat{\theta}_{jil} = \frac{n(x_j^{(i)}, x_{s(j)}^{(l)})}{n(x_{s(j)}^{(l)})}$$

where $n(x_{s(j)}^{(l)})$ is the number of times the configuration $x_{s(j)}^{(l)}$ appears in $\mathcal{D}$.

Proof. Recall from section 3.5.1 that the likelihood is given by
$L(\mathcal{D}|\mathbf{\Psi}) = \prod_{i=1}^{n} P_{X|\mathbf{\Psi}}(\mathbf{x}_{(k)}|\boldsymbol{\psi}, \mathcal{G}) = \prod_{j=1}^{d} \prod_{l=1}^{k_{s(j)}} \prod_{i=1}^{k_j} \theta_{jil}^{n(x_j^{(i)}, x_{s(j)}^{(l)})}$

This is simply $d \times k_{s(j)}$ separate maximum likelihood estimations all of the multinomial form discussed in section 3.6.1.

Hence

$$\hat{\theta}_{jil} = \frac{n(x_j^{(i)}, x_{s(j)}^{(l)})}{n(x_{s(j)}^{(l)})}.$$

Thus $\hat{\theta}_{jil} = \frac{\text{frequency of the family configuration}}{\text{frequency of the parent configuration}}$

□

3.6.2.1 Consistency of the MLE

In this section we will present some key notions and lemmas adapted from the work of Friedman & Koller [18] which will culminate in theorem 3.6.8 which states that the MLE is consistent.

Definition 3.6.1. Let P and Q be two distributions over $X_1, ..., X_d$. The relative entropy of P and Q is

$$\mathbf{D}(P(X_1, ..., X_d) \parallel Q(X_1, ..., X_d)) = \mathbb{E}_P \left[\log \frac{P(X_1, ..., X_d)}{Q(X_1, ..., X_d)} \right]$$

This measure is often known as the *Kullback-Liebler divergence*. It measures how much the distributions P and Q differ. It is clear that if $P = Q$ then $\mathbf{D}(P||Q) = 0$. Something which might not be evident at first glance, but straightforward to prove, is that the relative entropy is non-negative.

Lemma 3.6.3. For any two discrete probability distributions, it holds that

$$\mathbf{D}(P \parallel Q) \geq 0$$

Proof. The proof uses Jensen's inequality which states that for any random variable X and convex function $\phi, \mathbb{E}[\phi(X)] \geq \phi(\mathbb{E}[X])$. Hence, since -log is convex, we get that

$$D(P(\mathbf{X})||Q(\mathbf{X})) = -\sum_{\mathbf{X} \in \mathcal{X}} P(\mathbf{X}) \log \frac{Q(\mathbf{X})}{P(\mathbf{X})} \geq -\log \left(\sum_{\mathbf{X} \in \mathcal{X}} P(\mathbf{X}) \frac{Q(\mathbf{X})}{P(\mathbf{X})} \right) = -\log(1) = 0$$

where in the penultimate equality we use the fact that a discrete probability distribution sums to 1. □

Suppose Q is a distribution function which lies in a class of distributions $\mathcal{Q}$. $\mathcal{Q}$ may be, for example,a parametric family such as the Poisson family of distributions. $\mathcal{Q}$

is said to be a convex family of distributions if given $\lambda_1, ..., \lambda_r$ such that $\sum_{i=1}^{r} \lambda_i = 1$ and $\lambda_i \geq$ for $i = 1, ..., r$, and probability density functions $P_1, ..., P_r \in \mathcal{Q}$, then the PDF P defined by $P(x) = \sum_{i=1}^{r} \lambda_i P_i(x)$ also lies in $\mathcal{Q}$. Now, suppose we have a distribution P which we want to approximate using a distribution Q. A useful measure of such an approximation is given below.

Definition 3.6.2. (M-projection) Let P be a distribution and let $\mathcal{Q}$ be a convex set of distributions. The M-projection of P onto $\mathcal{Q}$ is the distribution

$$Q^M = \operatorname*{argmin}_{Q \in \mathcal{Q}} \mathbf{D}(Q \parallel P)$$

Lemma 3.6.4. For any distributions P and P' over the same state space $\mathcal{X}$

$$\mathbf{D}(P(\mathbf{X}) \parallel P'(\mathbf{X})) = -\mathbf{H}_P(\mathbf{X}) - \mathbb{E}_P[\log P'(\mathbf{X})]$$

where $\mathbf{H}_P(\mathbf{X})$ is the joint entropy of X defined in 3.5.8.

Proof.

$$\begin{aligned}
\mathbf{D}(P(\mathbf{X}) \parallel P'(\mathbf{X})) &= \mathbb{E}_P \left[\log \frac{P(\mathbf{X})}{P'(\mathbf{X})} \right] \\
&= \mathbb{E}_P \left[\log(P(\mathbf{X})) - \log(P'(\mathbf{X})) \right] \\
&= \mathbb{E}_P \left[\log(P(\mathbf{X})) \right] - \mathbb{E}_P \left[\log(P'(\mathbf{X})) \right] \\
&= -\mathbf{H}_P(\mathbf{X}) - \mathbb{E}_P \left[\log(P'(\mathbf{X})) \right]
\end{aligned}$$

□

Let P^* denote the PDF corresponding to the true distribution of $\mathbf{X}$. In general, we do not know P^* but we can estimate it using $\hat{P}_{\mathcal{D}}$. It turns out that this approximation has the following asymptotic property. The proof will not be covered here but can be found in a paper published by Berend & Kontorovich [5].

Theorem 3.6.5. $\lim_{n \to \infty} \hat{P}_{\mathcal{D}}(\mathbf{a}) = P^*(\mathbf{a})$ *almost surely.*

The following proposition shows that we can write the log-likelihood function in terms of the empirical distribution.

Proposition 3.6.6.

$$\log L(\mathcal{D}|\boldsymbol{\theta}) = n\mathbb{E}_{\hat{P}_{\mathcal{D}}}[\log P(\mathbf{X})]$$

Proof.

$$\begin{aligned}
\log L(\mathcal{D}|\boldsymbol{\theta}) &= \sum_{i=1}^{n} \log P_{X|\boldsymbol{\Psi}}(\mathbf{x}_{(i)}|\boldsymbol{\theta}) \\
&= \sum_{\xi \in \mathcal{X}} \left[\sum_{i=1}^{n} \mathbb{1}_{\{\mathbf{x}_{(i)}=\xi\}} \right] \log P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X} = \xi|\boldsymbol{\theta}) \\
&= \sum_{\xi \in \mathcal{X}} n\hat{P}_{\mathcal{D}}(\xi) \log P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X} = \xi|\boldsymbol{\theta}) \\
&= n\mathbb{E}_{\hat{P}_{\mathcal{D}}}[\log P(\mathbf{X})]
\end{aligned}$$

□

We are now able to derive one of the main results of this section.

Theorem 3.6.7. *The MLE $\hat{\boldsymbol{\theta}}$ in a parametric family is the M-projection onto the parametric family. That is*

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta} \in \boldsymbol{\Psi}} \mathbf{D}(\hat{P}_{\mathcal{D}} \parallel P_{\boldsymbol{\theta}})$$

Proof. By proposition 3.6.6 we know that $\log L(\mathcal{D}|\boldsymbol{\theta}) = n\mathbb{E}_{\hat{P}_{\mathcal{D}}}[\log P(\mathbf{X})]$. Applying lemma 3.6.4 we get that

$$\log L(\mathcal{D}|\boldsymbol{\theta}) = n\left(-\mathbf{H}_{\hat{P}_{\mathcal{D}}}(\mathbf{X}) - \mathbf{D}(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P(\mathbf{X})\right)$$

Since $\hat{\boldsymbol{\theta}}$ maximises the LHS and $\mathbf{D}(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P(\mathbf{X})) \geq 0$ it follows that it must also minimize the latter.

□

Thus we see that the MLE finds the distribution in the parametric family which is closest to the empirical distribution in terms of relative entropy. Combining this result with theorem 3.6.5 immediately gives us the following important result.

Theorem 3.6.8. *Given $P^*(\mathbf{X}) = P^*(\mathbf{X} : \boldsymbol{\theta}^*)$, that is, P^* is representable in the parametric family, then*

$$\hat{\boldsymbol{\theta}} \to \boldsymbol{\theta}^* \text{ almost surely as } n \to \infty.$$

Theorem 3.6.9. *(Strong Consistency) Let P^* be the PDF of the true distribution and let $\{P(\cdot|\boldsymbol{\theta}) : \theta \in \boldsymbol{\Phi}\}$ be a parametric family of distributions, and let $\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \mathbf{D}(P^*(\mathbf{X}) \parallel P(\cdot|\boldsymbol{\theta}))$ be the M-projection of P^* onto this family. Then*

$$\lim_{n\to\infty} P(\cdot|\hat{\boldsymbol{\theta}}) = P(\cdot|\boldsymbol{\theta}^*) \text{ almost surely.}$$

Before we go ahead with the proof of this theorem, it is useful to clearly articulate its implication. In words, it says that the distribution in the parametric family with parameter $\hat{\boldsymbol{\theta}}$ converges a.s to the distribution in the parametric family with true parameter $\boldsymbol{\theta}^*$.

Proof.

$$\begin{aligned}
\log \frac{P(\mathcal{D}|\hat{\boldsymbol{\theta}})}{P(\mathcal{D}|\boldsymbol{\theta})} &= \log P(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \log P(\mathcal{D}|\boldsymbol{\theta}) \\
&= n\left(-\mathbf{H}_{\hat{P}_{\mathcal{D}}}(\mathbf{X}) - \mathbf{D}\left(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\hat{\boldsymbol{\theta}}\right)\right) \\
&- n\left(-\mathbf{H}_{\hat{P}_{\mathcal{D}}}(\mathbf{X}) - \mathbf{D}\left(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta})\right)\right) \text{ (using 3.6.7)} \\
&= n\left(\mathbf{D}\left(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta})\right) - \mathbf{D}\left(\hat{P}_{\mathcal{D}}(\mathbf{X}) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\hat{\boldsymbol{\theta}})\right)\right) \\
&\approx n\left(\mathbf{D}\left(P^*_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta}^*) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta})\right) - \mathbf{D}\left(P^*_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta}^*) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\hat{\boldsymbol{\theta}})\right)\right) \\
&\quad (\text{since } \hat{P}_{\mathcal{D}} \to P^* a.s.)
\end{aligned}$$

Now, setting $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ we get that

$$\log \frac{P(\mathcal{D}|\hat{\boldsymbol{\theta}})}{P(\mathcal{D}|\boldsymbol{\theta}^*)} \approx n\left(\mathbf{D}\left(P^*_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta}^*) \parallel P^*_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta}^*)\right) - \mathbf{D}\left(P^*_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\boldsymbol{\theta}^*) \parallel P_{\mathbf{X}|\boldsymbol{\Psi}}(\mathbf{X}|\hat{\boldsymbol{\theta}})\right)\right)$$

where the first term in the brackets is equal to zero by defintion of $\mathbf{D}$ and the second term tends to zero as $n \to \infty$ since $\hat{\boldsymbol{\theta}} \to \boldsymbol{\theta}^*$. Note that both terms depend only on n and not on $\hat{P}_{\mathcal{D}}$.

Thus, when n is large, $\log \frac{P(\mathcal{D}|\hat{\boldsymbol{\theta}})}{P(\mathcal{D}|\boldsymbol{\theta}^*)} \approx 0$ which implies that $P(\cdot|\hat{\boldsymbol{\theta}})$ tends to $P(\cdot|\boldsymbol{\theta}^*)$ almost surely. □

3.6.3 Bayesian Parameter Estimation

In this section we will delve into the Bayesian approach for estimating the parameters of the model. The Bayesian approach allows one to incorporate any prior knowledge they might have on the parameters.

Let $\boldsymbol{\theta}_{j.l} = (\theta_{j1l}, ..., \theta_{jk_jl})$ be a vector of parameters. The prior distribution over $\boldsymbol{\theta}_{j.l}$ is taken to be $Dirichlet(\alpha_{j1l}, ..., \alpha_{jk_jl})$, where
$\alpha_{j1l}, ..., \alpha_{jk_jl} > 0$. In 3.6.11, we will see how these parameters can be chosen in a manner which incorporates any prior knowledge one may have of the conditional probabilities. A direct advantage the Dirichlet prior yields is that, as we will see in lemmma 3.6.10, it is relatively easy to obtain the distribution of the posterior density. The pdf is given by

$$\pi(\boldsymbol{\theta}_{j.l}) = \frac{\Gamma(\alpha_{j1l} + \alpha_{j2l} + ... + \alpha_{jk_jl})}{\Gamma(\alpha_{j1l})\Gamma(\alpha_{j2l})...\Gamma(\alpha_{jk_jl})}\theta_{j1l}^{\alpha_{j1l}-1}\theta_{j2l}^{\alpha_{j2l}-1}...\theta_{jk_jl}^{\alpha_{jk_jl}-1}$$

Lemma 3.6.10. Given that $\boldsymbol{\theta}_{j.l} \sim Dirichlet(\alpha_{j1l}, ..., \alpha_{jk_jl})$, the posterior distribution $\boldsymbol{\theta}_{j.l}|\mathcal{D}$, is given by

$$\boldsymbol{\theta}_{j.l}|\mathcal{D} \sim Dirichlet(\alpha^*_{j1l}, ..., \alpha^*_{jk_jl})$$

where $\alpha^*_{jil} = n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil}$.

Proof. We have that

$$\pi(\boldsymbol{\theta}_{j.l}) \propto \prod_{i=1}^{k_j} \theta_{jil}^{\alpha_{jil}-1}$$

and therefore by section 3.6.2

$$\begin{aligned}
\pi(\boldsymbol{\theta}_{j.l}|\mathcal{D}) &\propto \pi(\boldsymbol{\theta}_{j.l})L(\mathcal{D}|\boldsymbol{\theta}_{j.l}) \\
&\propto \pi(\boldsymbol{\theta}_{j.l}) \prod_{i=1}^{k_j} \theta_{jil}^{n(x_j^{(i)}, x_{s(j)}^{(l)})} \\
&\propto \theta_{j1l}^{\alpha_{j1l}-1}\theta_{j2l}^{\alpha_{j2l}-1}...\theta_{jk_jl}^{\alpha_{jk_jl}-1} \prod_{i=1}^{k_j} \theta_{jil}^{n(x_j^{(i)}, x_{s(j)}^{(l)})} \\
&\propto \prod_{i=1}^{k_j} \theta_{jil}^{\alpha^*_{jil}-1}
\end{aligned}$$

Thus the posterior density is proportional to $Dirichlet(\alpha^*_{j1l}, ..., \alpha^*_{jk_jl})$.
But since a density must integrate to 1, this implies that the posterior density must be the aforementioned Dirichlet distribution, that is

$$\boldsymbol{\theta}_{j.l}|\mathcal{D} \sim Dirichlet(\alpha^*_{j1l}, ..., \alpha^*_{jk_jl}) = Dirichlet(n(x_j^{(1)}|x_{s(j)}^{(l)})+\alpha_{j1l}, ..., n(x_j^{(k_j)}|x_{s(j)}^{(l)})+\alpha_{jk_jl})$$

□

We now use the posterior density derived above to obtain the predictive distribution of a new observation $\mathbf{x}_{(n+1)}$. We will estimate θ_{jil} by

$$\tilde{\theta}_{jil} = P(\{X_{n+1,j} = x_j^{(i)}\}|\{\Pi_j = x_{s(j)}^{(l)}\}, \mathcal{D}).$$

This is the predictive conditional distribution that the variable X_j attains value $x_j^{(i)}$ given that the parent configuration is $x_{s(j)}^{(l)}$ and data $\mathcal{D}$.

Lemma 3.6.11. Let $\alpha_{j.l} = \sum_{m=1}^{k_j} \alpha_{jml}$. Given the DAG $\mathcal{G}$ and data set $\mathcal{D}$,

$$\tilde{\theta}_{jil} = \frac{n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil}}{n(x_{s(j)}^{(l)}) + \alpha_{j.l}}$$

Before we state the proof of this lemma, it is useful to explain its implications. The lemma demonstrates that by taking the prior parameter α_{jil} to be relatively larger than the rest of the prior parameters $\{\alpha_{jml}|m \neq i\}$, we would be increasing the posterior estimate $\tilde{\theta}_{jil}$ of θ_{jil}. Thus, any prior information we might have on θ_{jil} can be incorporarated in the estimation by choosing α_{jil} accordingly.

Proof. Define

$$S_{jl} = \left\{ (\theta_{jil})_{i=1}^{k_j} | \theta_{jil} >= 0, i = 1, ..., k_j, \sum_{i=1}^{k_j} \theta_{jil} = 1 \right\}$$

Then

$$\begin{aligned}
\tilde{\theta}_{jil} &= \mathbb{P}\left(\{X_{n+1,j} = x_j^{(i)}\}|\{\Pi_j = x_{s(j)}^{(l)}\}, \mathcal{D}\right) \\
&= \int_{S_{jl}} \mathbb{P}\left(\{X_{n+1,j} = x_j^{(i)}\}|\{\boldsymbol{\theta}_{j.l} = (\theta_{j1l}, ..., \theta_{jk_jl})\}\right) \pi(\boldsymbol{\theta_{j.l}}|\{\Pi_j = x_{s(j)}^{(l)}\}, \mathcal{D}) d\boldsymbol{\theta}_{j.l} \\
&= \int_{S_{jl}} \theta_{jil} \pi(\boldsymbol{\theta}_{j.l}|\mathcal{D}, \alpha_{jil}) d\boldsymbol{\theta}_{j.l} \\
&= \int_{S_{jl}} \theta_{jil} \frac{\Gamma(\sum_{i=1}^{k_j} n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil})}{\prod_{m=1}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})} \prod_{m=1}^{k_j} \theta_{jml}^{n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml} - 1} d\boldsymbol{\theta}_{j.l} \\
&= \frac{\Gamma(\sum_{i=1}^{k_j} n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil})}{\prod_{m=1}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})} \int_{S_{jl}} \theta_{jil}^{n(x_j^{(i)}|x_{s(j)}^{(l)}) + \alpha_{jil}} \prod_{m=1, m\neq i}^{k_j} \theta_{jml}^{n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml} - 1} d\boldsymbol{\theta}_{j.l}
\end{aligned}$$

Now, note that

$$\theta_{jil}^{n(x_j^{(i)}|x_{s(j)}^{(l)}) + \alpha_{jil}} \prod_{m=1, m\neq i}^{k_j} \theta_{jml}^{n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml} - 1} \propto Dirichlet(\alpha_{j1l}, ..., \alpha_{jil} + 1, ...\alpha_{jk_jl})$$

and therefore

$$\int_{S_{jl}} \theta_{jil}^{n(x_j^{(i)}|x_{s(j)}^{(l)}) + \alpha_{jil}} \prod_{m=1, m\neq i}^{k_j} \theta_{jml}^{n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml} - 1} d\boldsymbol{\theta}_{j.l} =$$

$$\frac{\Gamma(n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil} + 1) \prod_{m=1, m\neq i}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})}{\Gamma(n(x_{s(j)}^{(l)}) + \alpha_{j.l} + 1)}$$

Hence, using the fact that $\Gamma(x+1) = x\Gamma(x)$ we get

$$\tilde{\theta}_{jil} = \frac{\Gamma(n(x_{s(j)}^{(l)}) + \alpha_{j.l})}{\prod_{m=1}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})}$$
$$\times \frac{\Gamma(n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil} + 1) \prod_{m=1, m\neq i}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})}{\Gamma(n(x_{s(j)}^{(l)}) + \alpha_{j.l} + 1)}$$

$$= \frac{\Gamma(n(x_{s(j)}^{(l)}) + \alpha_{j.l})}{\prod_{m=1}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})} \frac{(n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil}) \prod_{m=1}^{k_j} \Gamma(n(x_j^{(m)}|x_{s(j)}^{(l)}) + \alpha_{jml})}{(n(x_{s(j)}^{(l)}) + \alpha_{j.l})\Gamma(n(x_{s(j)}^{(l)}) + \alpha_{j.l})}$$

$$= \frac{n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha_{jil}}{n(x_{s(j)}^{(l)}) + \alpha_{j.l}}$$

□

Remark 3.6.1. *Note that if all the Dirichlet parameters are set to some constant α, then*

$$\tilde{\theta}_{jil} = \frac{n(x_j^{(i)}, x_{s(j)}^{(l)}) + \alpha}{n(x_{s(j)}^{(l)}) + k_j\alpha}$$

Thus if $\alpha = 0$, the Bayesian estimate $\tilde{\theta}_{jil}$ is equivalent to the MLE estimate $\hat{\theta}_{jil}$.

Let

$$T_{j.l}(\mathcal{D}) = \left(n(x_j^{(1)}|x_{s(j)}^{(l)}), ..., n(x_j^{(k_j)}|x_{s(j)}^{(l)})\right)$$

Lemma 3.6.12. $T_{j.l}(\mathcal{D})$ is a Bayesian sufficient statistic for $\boldsymbol{\theta}_{j.l}$.

Proof. Note that $T_{j.l}(\mathcal{D})$ contains all the relevant information for learning about $\boldsymbol{\theta_{j.l}}$ based on $\mathbf{X}$. Since we showed that the posterior distribution depends on $\mathbf{X}$ only through $T_{j.l}(\mathcal{D})$, we have proved the result.

□

Chapter 4

Tree-Based Methods

In this chapter we shall shift our focus from Bayesian networks to tree-based methods,namely random forest and boosting . We will first define what a Decision Tree is and how its structure is built from data. We then delve into random forests in section 4.2 which, as the name implies, is a collection of trees. Finally, in section 4.3 we delve into another tree based machine learning approach called *boosting*. We will be presenting these methods in the context of binary classification since this is the focus of this dissertation. However, these tree based approaches can also be extended to multiclass classification which shall not be covered in this dissertation. The results in this chapter were adapted from multiple sources, including, but not limited to, the works of Breiman et al. [8] and Freund & Schapire [13].

4.1 Decision Trees

Consider the data set $\mathcal{Z}$ containing n independent observations. A decision tree structure is constructed by iteratively splitting subsets of $\mathcal{Z}$ into two descendant subsets, beginning with $\mathcal{Z}$ itself. First, $\mathcal{Z}$ is divided into two subsets t_2 and t_3 according to a split s_1 which imposes a condition on a variable $X_j \in \boldsymbol{X}$. For example, the split s_1 could be of the form $X_2 \in A$ where A is some subset of $\mathcal{X}_2$. Thus by splitting $\mathcal{Z}$ using s_1, we get $t_2 = \{(\mathbf{x}_{(i)}, c_{(i)}) \in \mathcal{Z} : x_{i2} \in A\}$ and $t_3 = \{(\mathbf{x}_{(i)}, c_{(i)}) \in \mathcal{Z} : x_{i2} \notin A\}$. Similarly we then split t_2 into two subsets according to a split condition s_2 and we split t_3 according to s_3. This procedure is repeated until we reach a

stopping criterion (which will be defined and explained later on in 4.1.1) and hence stop splitting subsets. Suppose after splitting t_2 into subsets t_4 and t_5 using split condition s_2 and t_3 into t_6 and t_7 using s_3 we stopped splitting subsets. We could represent the previously described procedure using the tree like structure given in figure 4.1.1. Note that we will let $t_1 = \mathcal{Z}$.

Consider the tree given in figure 4.1.1. Note that each node t_i where $i = 1, ..., 7$ of the tree is a subset of $\mathcal{Z}$. There are three different kinds of nodes in a decision tree, namely:

- root node (t_1) - this node contains all of the training data $\mathcal{Z}$.
- terminal nodes - these are the leaf nodes of the tree and contain the final subsets of $\mathcal{Z}$ which are not subdivided further. The terminal nodes in figure 4.1.1 are t_4, t_5, t_6 and t_7. The terminal nodes form a partition of $\mathcal{Z}$.
- internal nodes - these nodes contain subsets not in the root node or in any of the terminal nodes, such as t_2 and t_3 in figure 4.1.1.

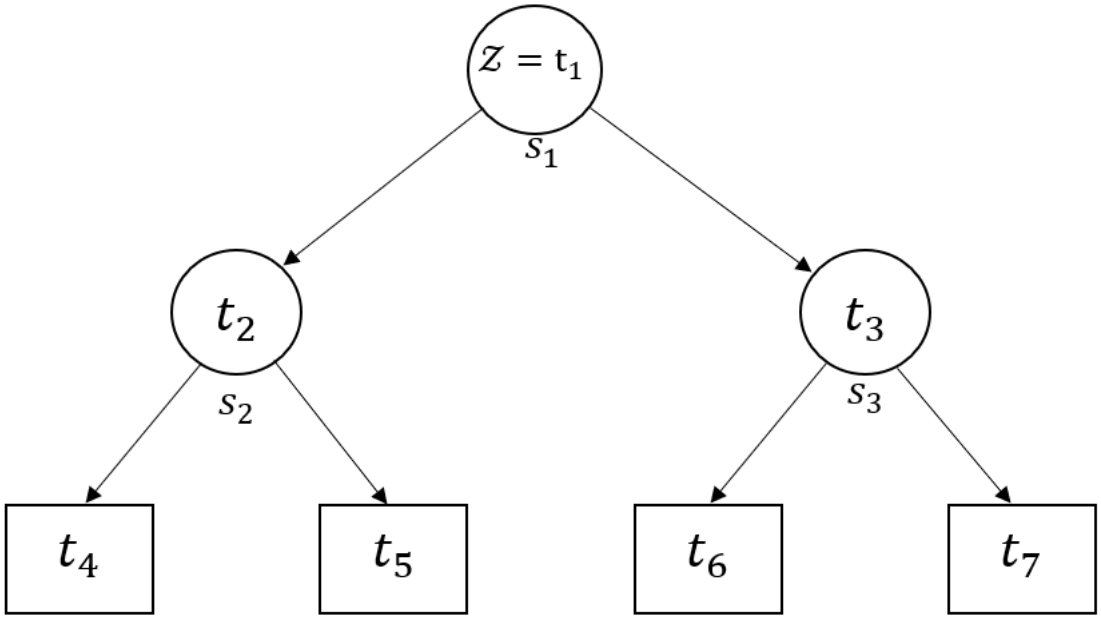

Figure 4.1.1: Decision Tree Classifier

We will now discuss how the tree classifier predicts the class label for an observation $\mathbf{x} = (x_1, ..., x_d)$ of $\mathbf{X}$ using the decision tree classifier in 4.1.1. First, suppose that the split conditions s_1, s_2 and s_3 are as follows:

- s_1: $X_2 \in A$ where $A \subset \mathcal{X}_2$
- s_2: $X_3 \in B$ where $B \subset \mathcal{X}_3$
- s_3: $X_d \in C$ where $C \subset \mathcal{X}_d$

From the definition of the first split s_1, it is determined whether $\mathbf{x}$ goes into t_2 or t_3. Suppose $x_2 \in A$ and hence $\mathbf{x}$ goes into t_2. We then use split s_2 to determine whether $\mathbf{x}$ goes into t_4 or t_5 by checking if $x_3 \in B$. Suppose the latter holds. Therefore $\mathbf{x}$ is placed in the terminal node t_4. Now, since our class variable C is binary, the predicted class of $\mathbf{x}$ will be the class which has majority representation in t_4.

4.1.1 Construction of the tree classifier

In the previous section we briefly discussed how a decision tree classifier partitions the data $\mathcal{Z}$ into subsets through the use of splits. However, we have not yet addressed the following two elements of the construction of the decision tree.

- The selection of the splits.
- How to decide whether to continue to splitting a node or declare it an internal node.

There are many methodologies available for dealing with the issues above when constructing decision tree models, such as the ID3 algorithm outlined by Quinlan [31], however we will cover the CART (Classification and Regression Tree) algorithm proposed by Breimann [8] due to its prevalence in the research we came across, such as that conducted by Philander [29] . We will now familiarise ourselves with the notions that are necessary to understand in order to fully comprehend the CART algorithm.

First of all, it is important to note that given a data set $\mathcal{Z}$, the set of all possible splits, denoted by S, is finite. This can be demonstrated as follows. Suppose

$X_j \in \mathcal{X}_j$ is a categorical variable with k_j possible values. A split s on X_j will be of the form $\{X_j \in A\}$ where A is a non-empty subset of $\mathcal{X}_j$. Since there are 2^{k_j-1} such subsets, there are as many ways to split an observation based on X_j. Let S_j be the set of all possible splits on $X_j, j = 1, .., d$. Moreover, let $S = S_1 \cup ... \cup S_d$. Since each S_i is finite, then so is S.

Let $p(j|t)$ where $j = 0, 1$ be the proportion of observations that belong to class j. We say that a subset of observations is *pure* if all the observations have the same class label, i.e $p(j|t) = 1$ for $j = 0$ or $j = 1$. When splitting a subset using some split condition, we want the data in each of the descendant subsets to be purer than the data in the parent subset.
A measure of the impurity of a node t is given by the *Gini Impurity Index* defined by

$$i(t) = \sum_{j \neq i} p(j|t)p(i|t) = 2p(1|t)p(0|t)$$

where the second equality comes from the fact that we are considering binary classification. Since $p(1|t) = (1 - p(0|t))$, we get that $i(t) = 2p(1|t)(1 - p(1|t))$ which, using differentiation, we can show has a maximum at $p(1|t) = \frac{1}{2}$. The same argument holds for $p(0|t)$. Thus if the observations in t are equally split amongst both classes, the impurity of the node will reach its maximal value. On the other hand, observe that if t is pure, then $i(t) = 0$.

Suppose there is a candidate split s which divides a node t into two nodes t_L and t_R such that a proportion p_L of the observations in t go into t_L and p_R go into t_R. Then the goodness of the split s is defined as the decrease in impurity given by

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R).$$

The larger this decrease in impurity is, the better the split s is. Having gone through the necessary fundamentals of decision tree construction, we are now in place to define the CART algorithm which is used to build a decision tree from a training data set $\mathcal{Z}$.

Algorithm 6: CART algorithm

Step 1: Generate the set S of all possible splits on the variables $X_1, ..., X_d$

Step 2: For a node t, find the split s^* which gives the largest decrease in Gini Impurity, i.e

$$\Delta i(s^*, t) = \max_{s \in S} \Delta i(s, t).$$

- If $\Delta i(s^*, t)$ is larger than some pre-established threshold δ, split t according to s^* and go back to step 1.
- If $\Delta i(s^*, t) < \delta$ make t a terminal node and assign observations in t to the class label $j = \underset{j=1,0}{\operatorname{argmax}}\, p(j|t)$.

Now that we have familiarised ourselves with the foundations of decision trees, as well as defined the CART algorithm, we will demonstrate an example based on a dummy data set.

Example 4.1.1. For demonstration purposes, assume we have obtained the dummy data set given in table 4.1 where

- X_1 is a binary variable which denotes whether a gambler has a total deposit count greater than 10 ($X_1 = 1$) or not ($X_1 = 0$).
- X_2 is a categorical variable which denotes whether a gambler's nationality is A,B or C.
- C is the binary classification variable which encodes whether a gambler is a problem gambler ($C = 1$) or not ($C = 0$).

By using the **rpart** package in RStudio, we applied the CART algorithm to the data set above and obtained the decision tree in figure 4.1.2.

Thus, this decision tree classifies an individual who has deposited more than ten times as a problem gambler, irrelevant of his/her nationality. If the gambler has deposited less than 10 times and his nationality is B, then the player is classified as a problem gambler, whilst if his nationality is either A or C then he is classified as a non problem gambler.

X_1	X_2	C
1	A	1
1	B	1
0	B	0
0	C	1
1	C	1

Table 4.1: Fictitious data on which CART algorithm is implemented.

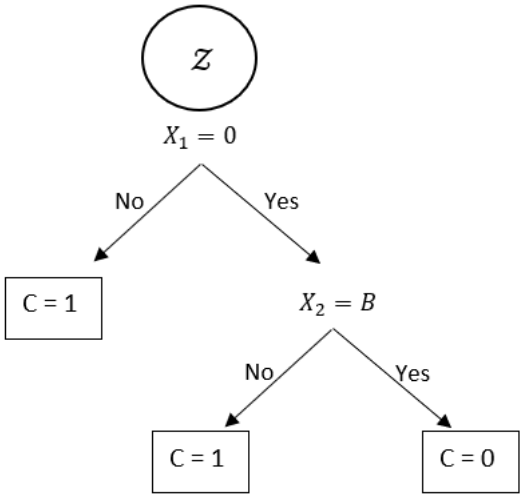

Figure 4.1.2: Final tree obtained using CART algorithm

4.2 Random Forests

With decision trees defined, we now proceed by delving into Random forests and how these models are built from a given data set. We first give the rigorous definition of a Random Forest in definition 4.2.1. Beforehand, we introduce the concept of bagging which is critical in the formulation of Random forests. Note that a decision tree classifier can be characterised by a function $h(\cdot, \boldsymbol{\Theta}) : \mathbf{x} \rightarrow c_i$ which maps an input vector $\mathbf{x}$ to a class label c_i given the split conditions in $\boldsymbol{\Theta}$.

The concept of Bagging (bootstrap aggregation) in terms of decision tree classifiers is as follows. From a sample of observations $\mathcal{Z}$, we select B bootstrap samples and fit a decision tree to each of these samples $\mathcal{Z}_1, ..., \mathcal{Z}_B$, each characterised by a function $h(\cdot, \mathbf{\Theta}_1), ..., h(\cdot, \mathbf{\Theta}_B)$ respectively, where Θ_j represents all the random choices we make when fitting the j^{th} tree. We then obtain a prediction for an input vector $\mathbf{x}$ by selecting the predominant class from the predictions $h(\mathbf{x}, \mathbf{\Theta}_1), ..., h(\mathbf{x}, \mathbf{\Theta}_B)$.

Definition 4.2.1. A *Random Forest* is a classifier consisting of a collection of decision tree classifiers $\{h(\mathbf{x}, \mathbf{\Theta}_k), k = 1, ...B\}$, where the $\{\mathbf{\Theta}_k\}$ are independent and identically distributed random vectors containing the splits and each tree casts a unit vote for the most popular class at input $\mathbf{x}$.

In order to present the algorithm proposed by Breiman [8] for building random forests, we first need to introduce a slight modification of the CART algorithm. The latter will be used to build each decision tree in the random forest and its steps are presented in the algorithm below. The motive behind this modification will be discussed later on in this section.

Algorithm 7: Modification of CART algorithm

Step 1: Select a random subset M of size m_{try} of the variables $X_1, ..., X_2$.

Step 2: Generate the set S' of all possible splits on the variables in M.

Step 3: For a node t, find the split s^* which gives the largest decrease in Gini Impurity, i.e

$$\Delta i(s^*, t) = \max_{s \in S'} \Delta i(s, t).$$

- If $\Delta i(s^*, t)$ is larger than some pre-established threshold δ, split t according to s^* and go back to step 1.
- If $\Delta i(s^*, t) < \delta$ make t a terminal node and assign observations in t to the class label $j = \underset{j=1,0}{\operatorname{argmax}}\, p(j|t)$.

Note that the main difference between algorithms 6 and 7 is that in the former, each time a split is to be performed, we perform an extensive search for the variable by computing the Gini Impurity for every single variable and choosing the one which

minimises this value. Contrarily, in algorithm 7, for each split to be performed, the search for the optimal splitting variable is limited to a subset of the variables available.

We now present the generalization of the algorithm proposed by Breiman for building a random forest from a training data set $\mathcal{Z}$.

Algorithm 8: Breiman's Random Forest algorithm

Given: a data set $\mathcal{Z}$ and a predetermined number of trees B.

for *i=1:B* **do**

Generate a bootstrap sample B_i of the data.

Grow a decision tree using algorithm 7.

end

Note that if $m_{try} = d$, then algorithm 8 is equivalent to bagging.

When performing bagging, the trees tend to make the same splits on the same variables meaning that the trees obtained are all similar looking. In this scenario, we say that the trees are correlated. When using random forests, since we use a different subset of variables for each split, a more diverse set of trees is produced which reduces the correlation present in bagging. As we shall see below, this has the advantage of decreasing the variance of the estimated class label given by the random forest classifier.

Let $Y_i(\mathbf{X}) = h(\mathbf{X}, \mathbf{\Theta}_i)$ be the output of decision tree i given input vector $\mathbf{X}$. Consider the B *iid* random variables $Y_1, ..., Y_B$, each with variance σ^2 and pairwise correlation ρ. Note that for binary classification where $C \in \{0, 1\}$, by definition of $h(\cdot, \mathbf{\Theta}_i)$,

$$Y_i(\mathbf{X}) = \begin{cases} 1 & \text{if predicted class is C} = 1 \\ 0 & \text{if predicted class is C} = 0 \end{cases}$$

Consider $\bar{Y}(\mathbf{X}) = \frac{1}{B}\sum_{i=1}^{B} Y_i(\mathbf{X})$. This may be viewed as the proportion of times, out of a total of B times, that $C = 1$ is predicted as the class label of $\mathbf{X}$.

Now,

$$\begin{aligned}\text{Var}\left(\bar{Y}(\mathbf{X})\right) &= \text{Var}\left(\frac{1}{B}\sum_{i=1}^{B} Y_i(\mathbf{X})\right) \\ &= \frac{1}{B^2}\sum_{i=1}^{B}\left(\sum_{j\neq i}\text{Cov}\left(Y_i(\mathbf{X}), Y_j(\mathbf{X})\right) + \text{Var}\left(Y_i(\mathbf{X})\right)\right) \\ &= \frac{1}{B^2}\left(B(B-1)\sigma^2\rho + B\sigma^2\right) \\ &= \rho\sigma^2 + \frac{\sigma^2(1-\rho)}{B}\end{aligned}$$

Thus, by increasing B and decreasing the absolute value of ρ using random forests, we decrease the variance of $\bar{Y}(\mathbf{X})$ which in turn means that we decrease the variance of the estimated class label. Since B can be increased by using bagging, the formula above explains why bagging reduces the variance when compared to using a single decision tree classifier. Moreover, it shows why random forests are superior to bagging in terms of variance since the trees in the latter are more correlated as we mentioned previously.

4.2.1 Theoretical Background on Random Forests

In this section we will introduce some theoretical concepts of random forests which culminate in theorem 4.2.1

Consider the decision tree classifiers $h(\mathbf{x}, \boldsymbol{\Theta}_1), h(\mathbf{x}, \boldsymbol{\Theta}_2), ..., h(\mathbf{x}, \boldsymbol{\Theta}_B)$. We define the margin function mg as follows.

$$mg(\mathbf{X}, C) = \frac{1}{B}\sum_{i=1}^{B}\mathbb{1}_{\{h(\mathbf{X},\boldsymbol{\Theta}_i)=C\}} - \max_{j\neq C}\left(\frac{1}{B}\sum_{i=1}^{B}\mathbb{1}_{\{h(\mathbf{X},\boldsymbol{\Theta}_i)=j\}}\right).$$

The margin measures the extent to which the average number votes at input $\mathbf{X}$ for the right class C exceeds the average vote of any other class. Clearly, the larger the margin is the more confident we are that the random forest has classified $\mathbf{X}$ correctly, and vice versa.

The generalization error, PE^*, of a random forest classifier is given by

$$PE^* = P_{\mathbf{X},C}(mg(\mathbf{X}, C) < 0).$$

In the following theorem we show that PE^* has a limiting value as we increase the number of decision trees in the random forest.

Theorem 4.2.1. *As the number of trees increases, PE^* converges almost surely to*

$$P_{\mathbf{X},C}\Big(P_{\mathbf{\Theta}}(h(\mathbf{x},\mathbf{\Theta})=C)-\max_{j\neq C}P_{\mathbf{\Theta}}(h(\mathbf{x},\mathbf{\Theta})=j)<0\Big) \tag{4.1}$$

Proof. Since we want to prove almost sure convergence, it suffices to prove that there is a set Z of probability zero on the sequence space $\mathbf{\Theta}_1, \mathbf{\Theta}_2, ...$ such that outside of Z, for all $\mathbf{x}$,

$$\frac{1}{B}\sum_{i=1}^{B}\mathbb{1}_{\{h(\mathbf{x},\mathbf{\Theta}_i)=j\}} \to P_{\mathbf{\Theta}}(h(\mathbf{x},\mathbf{\Theta})=j) \text{ as } B\to\infty.$$

For a fixed training set and fixed $\mathbf{\Theta}$, the set of all $\mathbf{x}$ such that $h(\mathbf{x},\mathbf{\Theta})=j$ is a union of hyper-rectangles.

For all $h(\mathbf{x},\mathbf{\Theta})$ there is only a finite number K of such unions of hyper-rectangles, denoted by $S_1, ..., S_K$. This may be seen as follows. Assume $\mathbf{x}\in\mathbb{R}^d$. Each split in $\mathbf{\Theta}$ corresponds to a partitioning the classification region $\mathbb{R}^d$. Therefore, since the no of splits is finite, so is the number of regions (hyper-rectangles) $\mathbf{\Theta}$ partitions $\mathbb{R}^d$ into. Hence there is only finitely many unions of hyper-rectangles.

Define

$$\phi(\mathbf{\Theta}) = \begin{cases} k & \text{if } \{\mathbf{x}: h(\mathbf{x},\mathbf{\Theta})=j\}=S_k, \\ 0 & \text{otherwise.} \end{cases}$$

Let $B_k := |\{\mathbf{\Theta}_i\in\mathbf{\Theta}_1, ..., \mathbf{\Theta}_B : \phi(\mathbf{\Theta}_i)=k\}|$. Hence B_k denotes the number of times that $\phi(\mathbf{\Theta}_i)=k$ in the first B trials. Then

$$\frac{1}{B}\sum_{i=1}^{B}\mathbb{1}_{\{h(\mathbf{x},\mathbf{\Theta}_i)=j\}} = \frac{1}{B}\sum_{k=1}^{K}B_k\mathbb{1}_{\{\mathbf{x}\in S_k\}}.$$

Note that we may rewrite B_k as $\sum_{i=1}^{B}\mathbb{1}_{\{\phi(\mathbf{\Theta}_i)=k\}}$ and, using the Strong Law of Large Numbers, it follows that $\frac{B_k}{B}$ converges a.s to $P_{\mathbf{\Theta}}(\phi(\mathbf{\Theta})=k)$. Taking unions of all the sets on which convergence does not occur for some value of k gives a set Z of probability zero such that for all $\mathbf{x}$ outside of Z,

$$\frac{1}{B}\sum_{i=1}^{B}\mathbb{1}_{\{h(\mathbf{x},\mathbf{\Theta}_i)=j\}} \xrightarrow{\text{P a.s}} \sum_{k=1}^{K}P_{\mathbf{\Theta}}(\phi(\mathbf{\Theta})=k)\mathbb{1}_{\{\mathbf{x}\in S_k\}}.$$

Finally, we note that the right hand side is precisely equal to $P_{\boldsymbol{\Theta}}(h(\mathbf{x}, \boldsymbol{\Theta}) = j)$ since $\sum_{k=1}^{K} P_{\boldsymbol{\Theta}}(\phi(\boldsymbol{\Theta}) = k)\mathbb{1}_{\{\mathbf{x} \in S_k\}} = \sum_{k=1}^{K} P_{\boldsymbol{\Theta}}(\{\mathbf{x} : h(\mathbf{x}, \boldsymbol{\Theta}) = j\} = S_k)\mathbb{1}_{\{\mathbf{x} \in S_k\}}$ □

4.3 Boosting

Another tree-based machine learning approach we will delve into is *Boosting*. Before we can delve into *Boosting*, we need to define what a *weak learner* is. A *weak learner* is a classification/regression machine learning algorithm which provides an accuracy slightly larger than that of random guessing. An example of a weak learner could be a decision tree with only 1 split. Such a decision tree is commonly referred to as a *stump*.

Boosting is a machine learning approach that is based on the idea of combining many different weak learners to produce a better prediction rule. The boosting algorithm we will discuss is the *AdaBoost* algorithm by Freund and Schapire [13] since it is one of the most widely used and studied. Before we delve into the Adaboost algorithm, we need to clarify some notions which are essential for the algorithm.

Suppose W is a discrete random variable taking values in $\{1, ..., n\}$. We write $I \sim W$ if I is a random variable which has the same probability distribution as that of W. Let i be an observed value of I. Therefore, $P_{I \sim W}(i = k) = P(W = k)$.

Now, suppose we have a weak learner h_t and a discrete distribution W_t also taking values in $\{1, ..., n\}$. We will denote $P(W_t = i)$ by $W_t(i)$. The *weighted error* of h_t is given by

$$\begin{aligned}
\epsilon_t &= P_{I \sim W_t}(h_t(\mathbf{x}_{(i)}) \neq c_{(i)}) \\
&= P_{I \sim W_t}(\mathbb{1}_{\{h_t(\mathbf{x}_{(i)}) \neq c_{(i)}\}} = 1) \\
&= \mathbb{E}_{I \sim W_t}[\mathbb{1}_{\{h_t(\mathbf{x}_{(i)}) \neq c_{(i)})\}}] \\
&= \sum_{i=1}^{n} \mathbb{1}_{h_t(\mathbf{x}_{(i)} \neq c_{(i)})} W_t(i)
\end{aligned}$$

The penultimate equality uses the fact that the probability of an indicator variable taking the value 1 is equal to its expectation. We now define the steps involved in

the implementation of the AdaBoost algorithm on a training data set $\mathcal{Z}$.

Algorithm 9: AdaBoost algorithm

Assume that we are given a data set $\mathcal{Z} = \big((\boldsymbol{x}_{(1)}, c_{(1)}), ..., (\boldsymbol{x}_{(n)}, c_{(n)})\big)$ containing n independent observations of $\boldsymbol{X} = (X_1, ..., X_d)$ and the class variable $C \in \{-1, +1\}$.

Initialize $W_1(i) = \frac{1}{n}$ for $i = 1, ..., n$. These are referred to as the sample weights.

for *t=1:T* **do**

Train weak learner h_t using distribution W_t.

Calculate the weighted error $\epsilon_t = P_{I\sim W_t}(h_t(\mathbf{x}_{(i)}) \neq c_{(i)}) = \sum_{i=1}^{n} W_t(i)\mathbb{1}_{\{h_t(\mathbf{x}_{(i)})\neq c_{(i)}\}}$.

Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.

for *i=1:n* **do**

$$\tilde{W}_{t+1}(i) = \begin{cases} W_t(i)\exp\left(-\alpha_t\right)) & \text{if } h_t(\mathbf{x}_{(i)}) = c_{(i)}, \\ W_t(i)\exp\left(\alpha_t\right)) & \text{if } h_t(\mathbf{x}_{(i)}) \neq c_{(i)}. \end{cases}$$

end

Compute the normalization factor $\mathrm{Z}_t = \sum_{i=1}^{n} \tilde{W}_{t+1}(i)$

Comment: we now update the sample weights

for *j=1:n* **do**

$\mathrm{W}_{t+1}(j) = \frac{\tilde{W}_{t+1}(j)}{Z_t}$

end

end

The final classification learner may be seen as a weighted combination of the weak learners given by

$$H(\mathbf{x}) = \mathrm{sign}(\sum_{i=1}^{T} \alpha_t h_t(\mathbf{x})).$$

The idea of AdaBoost is to iteratively fit weak learners which aim to improve the prediciton of previously misclassified observations by increasing their sample weights. Note that since α_t is large for small values of $\hat{\epsilon}_t$ and vice versa, a weak learner with small weighted error $\hat{\epsilon}_t$ will be given a higher weight α_t and thus have more influence on the final classification, and vice versa.

Chapter 5

Implementation

In this section we will be discussing the implementation of the models described in chapters 3 and 4. R software was used throughout the applications due to its vast array of off the shelf packages. In order to compare the models and evaluate their fit, we will compute various goodness of fit measures which shall be discussed in section 5.2. In section 5.3 we implement Bayesian Classifiers through the FSSJ algorithm as well as a modification of the CL-adapt algorithm. After having analysed the results obtained by these models, we then fit the Random Forest algorithm we discussed in chapter 3, namely algorithm 8 proposed by Breiman as well as AdaBoost. Note that throughout the sections mentioned, we will compare the measures of fit obtained on both the training data and the testing data in order to check whether there are indications of overfitting. Finally, in section 5.6 we will compare the algorithms with one another.

5.1 The Data

The data set used throughout this chapter was obtained from an online gaming company operating under an MGA license. It consists of 130588 observations with 22 predictor variables and 1 response variable. The data set was partitioned into a training set and testing set using a 75:25 ratio. The data includes 65389 users who self-excluded themselves from the gambling operator and 65199 users who never self-excluded. In this study, self-exclusion will be used as a proxy for problem gambling.

The latter technique was adopted by multiple studies mentioned in chapter 2. The variables related to transaction data such as deposits and money staked are based on a period of thirty days unless specified otherwise. The R package **bnclassify** which was used to fit Bayesian networks to our data set only handles discrete variables. Therefore, all continuous variables in the raw data set were discretized using the *discretizeDF* function found in the R package **arules**. The frequency method was used which ensures that there are an equal number of observations in each category. More information on this technique can be found in the aforementioned R package. Note that in the online gambling industry, the GGR (Gross Gaming Revenue) generated by a player is defined as the sum of the player's bets subtracted by the sum of their wins. A list of the variables used throughout this study, together with their description, may be found in table 5.1 below.

Variable Name	Description	Categories
category_segment	the type of games the user mostly plays	Slots,Slots\&Table,Table, Table\&Slots, Other
segment_value	the customer segment of the user based on segmentation performed by the company	SVP (Super Value Player), HVP (High Value Player), LVP (Low Value Player), MVP (Medium Value Player), SVP MAX, NV (No Value)
season	the season in which the user is active	autumn,spring,summer,winter
age_group	the age group of the user	18-20, 21-30, 31-40, 41-50, 51-60, 61+
brand	the brand of the company which the user engages with	Brand 1, Brand 2,Brand 3, Brand 4, Brand 5
country_code	the country code of the user's nationality	DE (Germany), FI (Finland), NL (Netherlands), ROW (Rest of World)
first_accepted_deposit_amount	the amount in euro of the first succesful deposit made by the user	[1,25), [25,50), [50,999]
log_total_deposit_amount	the logarithm of the total deposit amount made by the user in thirty days	[0,2.18), [2.18,2.85), [2.85,5.41]
log_total_money_bet_amount	the logarithm of the total amount of money staked	[0,2.87), [2.87,3.71), [3.71,7.65]
ggr_to_deposits_ratio	the ratio of deposits to ggr	[-726,0.228), [0.228,1), [1,251]
avg_stake	the average stake of a player	[0,0.48), [0.48,1.41), [1.41,998]
log_total_deposit_count	the logarithm of the total amount of times the user made a deposit	[0,0.415), [0.415,1.28), [1.28,998]
active_days	the amount of days the player was active	[0,2), [2,8), [8,31]
depositing_days	the amount of days the player placed at least one deposit	[0,2), [2,6), [6,31]
depositing_days_last_7_days	the amount of days within the last week of the time frame that the user deposited at least once	[0,1), [1,3), [3,8]
log_avg_deposit_amount_per_active_day	the logarithm of the average deposit amount per active day	[0,1.61), [1.61,2.09), [2.09,4.78]
log_avg_stake_per_active_day	the logarithm of the average amount the user staked per active day	[0,2.32), [2.32,2.92), [2.92,6.4]
log_avg_deposit_count_per_active_day	the logarithm of the average amount of deposits the user made per active day	[0,0.301), [0.301,0.477), [0.477,1.84]
winsorized_game_win_margin	the ratio of the amount of money won to the amount of money staked by the user	[-1,0.0306), [0.0306,0.164), [0.164,1]
login_duration_hours_month	the amount of hours the user was active within the time frame	[0,4.4), [4.4,19.8), [19.8,476]
login_duration_hours_week	the amount of hours the user was active within the last seven days of the time frame	[0,2.17), [2.17,8.28), [8.28,189]
problem_gambler	indicates whether a user self-excluded or not	TRUE,FALSE

Table 5.1: Description of Variables

5.2 Goodness of Fit Measures

In our application, we are keen to predict whether a gambler may be classified as a problem gambler or not. The following basic definitions are fundamental to the majority of the goodness of fit measures we will be using.

- True Positives (TP): The number of players who were classified as problem gamblers and are in reality problem gamblers.
- True Negatives (TN): The number of players who are not problem gamblers and were correctly classified as such.
- False Positives (FP): The number of players that were classified as problem gamblers but are in fact not problem gamblers.
- False Negatives (FN): The number of players that were not classified as problem gamblers but are in fact problem gamblers.

We are now in place to introduce the confusion matrix which is shown in figure 5.2.1 below.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	TN	FN
	Problem Gambler	FP	TP

Figure 5.2.1: Confusion matrix formulation

From the data given in the figure above we can derive a multitude of goodness of fit measures. The most sought after is the model's accuracy which is calculated as follows

$$Accuracy = \frac{\mathrm{TN} + \mathrm{TP}}{\mathrm{TN} + \mathrm{TP} + \mathrm{FN} + \mathrm{FP}}.$$

This is simply the proportion of observations whose predicted class was correct. Another metric commonly evaluated for binary classification is the model's sensitivity, also known as recall. The latter evaluates the model's ability to predict true positives. It is computed as

$$Sensitivity = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Similarly, the specificity of the model is defined as it's ability to predict true negatives. It is thus computed as

$$Specificity = \frac{\text{TN}}{\text{TN} + \text{FP}}.$$

Another relevant metric is a model's precision which is worked out as follows.

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The precision of a model is penalized whenever we predict a false positive whilst the sensitivity is penalized by false negatives. Ideally, in a good classifier we want both the precision and recall to be as close to one as possible. A metric which evaluates both quantities in conjunction is the $F1$ score given by

$$F1\ Score = 2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity}.$$

Note that the F1 score becomes one when both the precision and the sensitivity of the model are equal to one. In general, a high F1 Score is achieved by having a high precision as well as a high sensitivity.

We will now see how some of these measures of fit can be incorporated in one graphical representation called the *ROC curve*.

ROC/AUC

Let the class variable C take values in $\{0, 1\}$ and suppose we have a binary classifier which when given an observation $\mathbf{x}_{(i)}$, returns the probability p_i of the observation belonging to class $C = 1$. For a Bayesian classifier, $p_i = \hat{P}(C = 1|\mathbf{x}_{(i)})$. On the other hand,for a random forest with B decision trees, $p_i = \frac{1}{B}\sum_{j=1}^{B} \mathbb{1}_{\{h_j(\mathbf{x}_{(i)},\Theta)=1\}}$. This is equivalent to the proportion of trees which vote for $C = 1$.

Let t denote the threshold probability for an observation to be classified as belonging to class $C = 1$. This means that if $p_i > t$, the predicted class for $\mathbf{x}_{(i)}$ is $C = 1$, and $C = 0$ otherwise.

Now, suppose that for different values of t, say $t_1, t_2, ..., t_r$, we classify all the observations in the test data set and obtain r different confusion matrices. The ROC curve is then obtained as follows: For each different value of t, plot $1-$*Specificity* on the x-axis and the *Sensitivity* on the y-axis. The aim of the ROC graph is to summarize all the confusion matrices pertaining to different threshold values for the same classifier. Examples of the ROC curve and its interpretations will be provided in sections 5.3,5.4 and 5.6.

A diagnostic strongly related to the ROC curve is the area under the curve, commonly referred to as AUC. An AUC measure of 0 reflects that all values are misclassified whilst a value of 1 indicates a perfect classifier. An AUC of 0.5 is the result of an ROC curve which is equivalent to the line $y = x$. Such an ROC curve pertains to a model which is just as good as a random guessing. Ideally we would want our ROC curve to lie above $y = x$, indicating that the model has some predictive power compared to random guessing.

The ROC curve can be used to determine the "optimal" threshold value which minimizes misclassification. A technique commonly used is the *closest to (0,1) criteria* which is explained by Perkins et al [27]. This approach finds the threshold value which corresponds to the point on the ROC curve closest to the coordinate $(0, 1)$. More explicitly, if $\{(x_1, y_1), ..., (x_r, y_r)\}$ are the coordinates corresponding to $t_1, ..., t_r$ respectively, i.e threshold probability t_i yields $1 - Specificity = x_i$ and $Sensitivity = y_i$, then the index corresponding to the "optimal" threshold value is calculated as

$$i^* = \underset{i}{\operatorname{argmin}} \sqrt{(x_i)^2 + (1 - y_i)^2}.$$

The optimal threshold value is then given by t_{i^*}. We will be using the *coords* function pertaining to the **pROC** package to calculate this value.

5.3 Bayesian networks for Identifying Problem Gamblers

The package of choice for fitting the Bayesian networks discussed in sections 5.3.1 and 5.3.2 is **bnclassify** by Mihaljevic et al [23]. We shall now discuss the implementation and results obtained when using the aforementioned package to fit Bayesian networks to our data set.

5.3.1 Chow-Liu algorithm

As we mentioned earlier, the adaptation of the Chow-Liu algorithm which we referred to as CL-adapt in chapter 3 maximises the likelihood and returns a DAG which includes all the input variables as nodes. This means that the model may not be as parsimonious as one may desire and hence in this section we will introduce and discuss a further adaptation of the said algorithm which aims to return a Bayesian network with less nodes and minimal compromise in accuracy.
We shall now go over the pseudo code of the alteration of the CL-adapt algorithm which we implemented in R. Afterwards, an overview of the most important steps of the algorithm will be given.

Algorithm 10: Pseudo code for alteration of CL-adapt

```
split data set 𝒵 into training and testing data sets;
attributes ← list containing names of attributes X_1, ..., X_d;
models ← vector of size d; accuracy_vec ← vector of size d;
for i=1:d do
    diff ← 1; epsilon ← 0.005; starting_variable ← attributes(i);
    chosen_attributes ← dynamic vector;
    chosen_attributes.add(starting_variable);
    attributes_not_being_used ← list of all attributes apart from starting_variable;
    random_attribute ← sample 1 attribute at random from attributes_not_being_used;
    old_model ← fit CL-adapt to training data using only random_attribute and
     starting_variable as features;
    old_accuracy ← accuracy(old_model);
    current_accuracy ← old_accuracy;
    while diff ≥ epsilon do
        for attribute in attributes_not_being_used do
            new_bn ← fit CL-adapt to training data using features in chosen_attributes and
             attribute;
            new_accuracy ← accuracy(new_bn);
            if new_accuracy ≥ current_accuracy then
                current_accuracy ← new_accuracy;
                new_model ← new_bn; current_best_attribute ← attribute;
            end
        end
        diff ← current_accuracy - old_accuracy;
        if diff ≥ epsilon then
            old_bn ← new_model;
            old_accuracy ← current_accuracy;
            attributes_not_being_used.remove(current_best_attribute);
             chosen_attributes.add(current_best_attribute)
        else
            chosen_model ← old_bn;
            models.add(chosen_model);
            accuracy_vec.add(old_accuracy)
        end
    end
end
```

The algorithm chooses a starting variable and then fits a Bayesian network together with this starting variable and another variable chosen at random using the CL-adapt algorithm. It then loops through the rest of the variables one by one to check which one increases the accuracy the most when fitted together with the starting variable. The variable current_best_attribute which leads to the biggest increase in accuracy is added to the model. This process is then repeated until there is no longer any significant increase (at least epsilon) when adding an attribute to the model. The final model is then stored in the vector of models and we start the process over again using a different starting variable.
After we have run the process using all different attributes as starting variables, we then compare the accuracy of all the models in the models vector and return the model corresponding to the greatest accuracy.

The *bnc* function is used to implement the CL-adapt algorithm at each iteration of algorithm 10. The said function takes as input a parameter α which indicates whether MLE or Bayesian estimation will be used to estimate the conditional probabilities defined in 3.2. Setting $\alpha = 0$ indicates that maximum likelihood estimation will be used, whilst setting α to be any real number greater than 0 indicates that Bayesian parameter estimation will be used with all the Dirichlet parameters set to α. As discussed previously in section 3.6.3, using Bayesian parameter estimation with all Dirichlet parameters equal to a constant does not allow one to incorporate prior information. Hence in our implementation, we set $\alpha = 0$ in order to use maximum likelihood estimation.

The confusion matrix in figure 5.3.1 was returned when we fitted this model on the training data set. The obtained accuracy is 0.7213, along with a specificity of 0.7444 and a sensitivity of 0.6983. The F1 score is given by 0.7152.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	36373	14807
	Problem Gambler	12492	34269

Figure 5.3.1: Confusion matrix of alteration of CL-adapt when fitted on training set

When using $t = 0.5$ as a threshold probability, the accuracy of this model on the test data set was 0.7200355. This value is immensely close to the accuracy obtained on the training set, indicating that overfitting is likely not present in this model. From the confusion matrix given in figure 5.3.2, we get that the sensitivity of the model, which evaluates its ability to predict true positives of each available category, is given by 0.6769. The specificity for the model in consideration is 0.7464. Hence, the model's ability to correctly predict that someone is not a problem gambler is superior to its ability to detect a problem gambler. Moreover, the precision is 0.732 and therefore the F1 score is given by 0.7034.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	12192	4998
	Problem Gambler	4142	11315

Figure 5.3.2: Confusion matrix of alteration of CL-adapt on testing set

We also fitted the CL-adapt algorithm (3) on the full-set of variables in order to compare its accuracy compared to the model returned by algorithm 10. The model returned by the former algorithm which uses all 23 predictor variables returned an accuracy of 0.729531 which is only marginally better than that returned by the latter algorithm which uses 5 predictor variables. Thus, as we alluded to in the introduction of this section, the BN returned by algorithm 10 is more parsimonious and has almost equivalent predictive power compared to the BN returned when applying algorithm 3 on the full set of predictor variables.

When implementing algorithm 10 in R, the Bayesian network seen in figure 5.3.3 was returned. The variables present in the DAG are those which the algorithm deemed relevant in order to predict problem gambling. The DAG suggests, for example, that whether a player is a problem gambler or not will have an influence on the amount of days said player is active on the site as well as on his/her segment value. From

the DAG we can derive conditional irrelevance statements regarding the variables. Using the Local Markov Property defined in definition 3.3.2, we can deduce that the *log_avg_deposit_count_per_active_day* of a player is independent of the set of variables {*active_days,country_code,segment_value*} given that we have information on the variable set {*problem_gambler,log_avg_stake_per_active_day*}. However, in practice, such a statement is not particularly useful since we would not know beforehand whether a player is a problematic gambler or not.

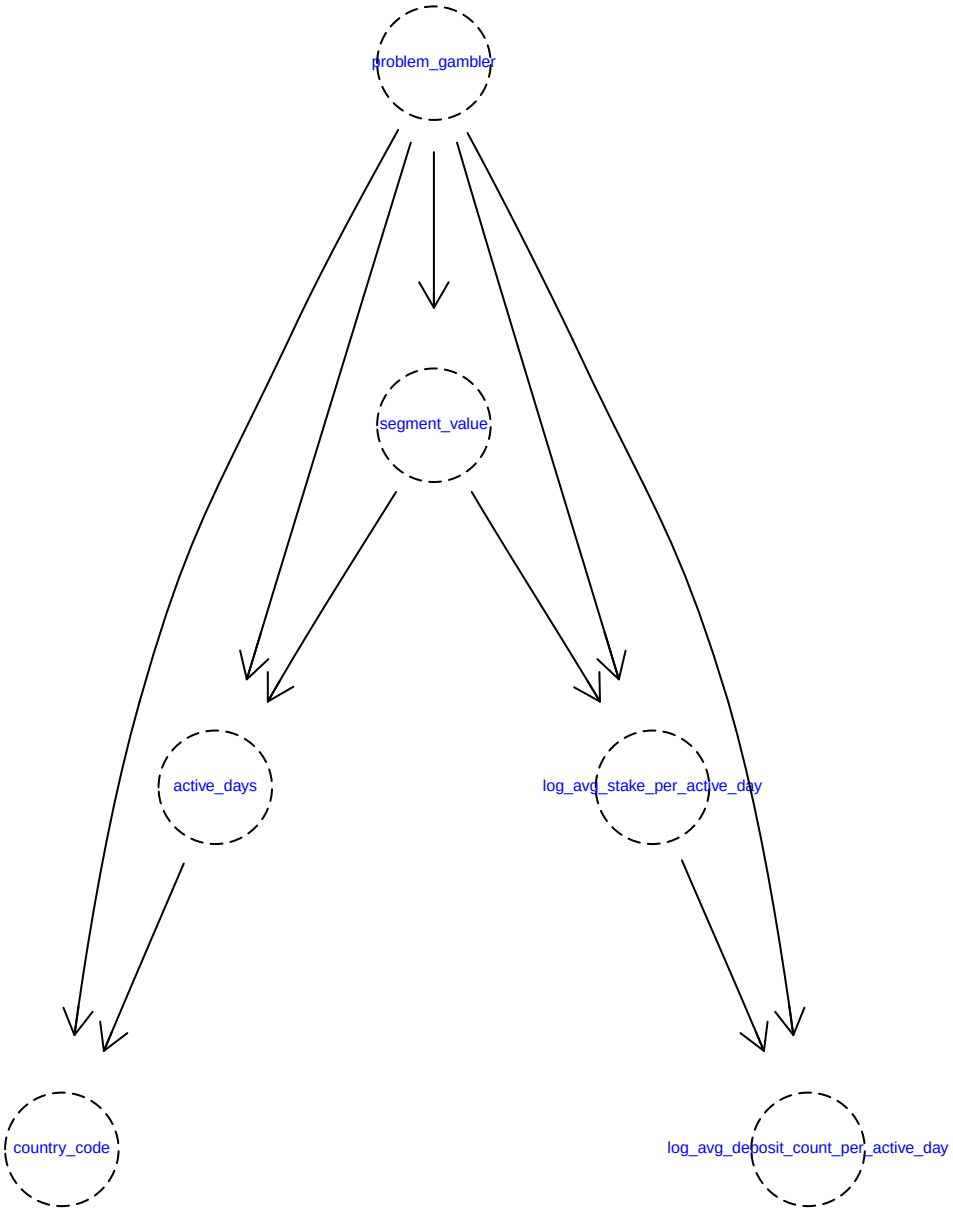

Figure 5.3.3: Bayesian network returned from alteration of CL-adapt implemented in R

By using the **pROC** package in R we obtained the ROC curve shown in figure 5.3.4.

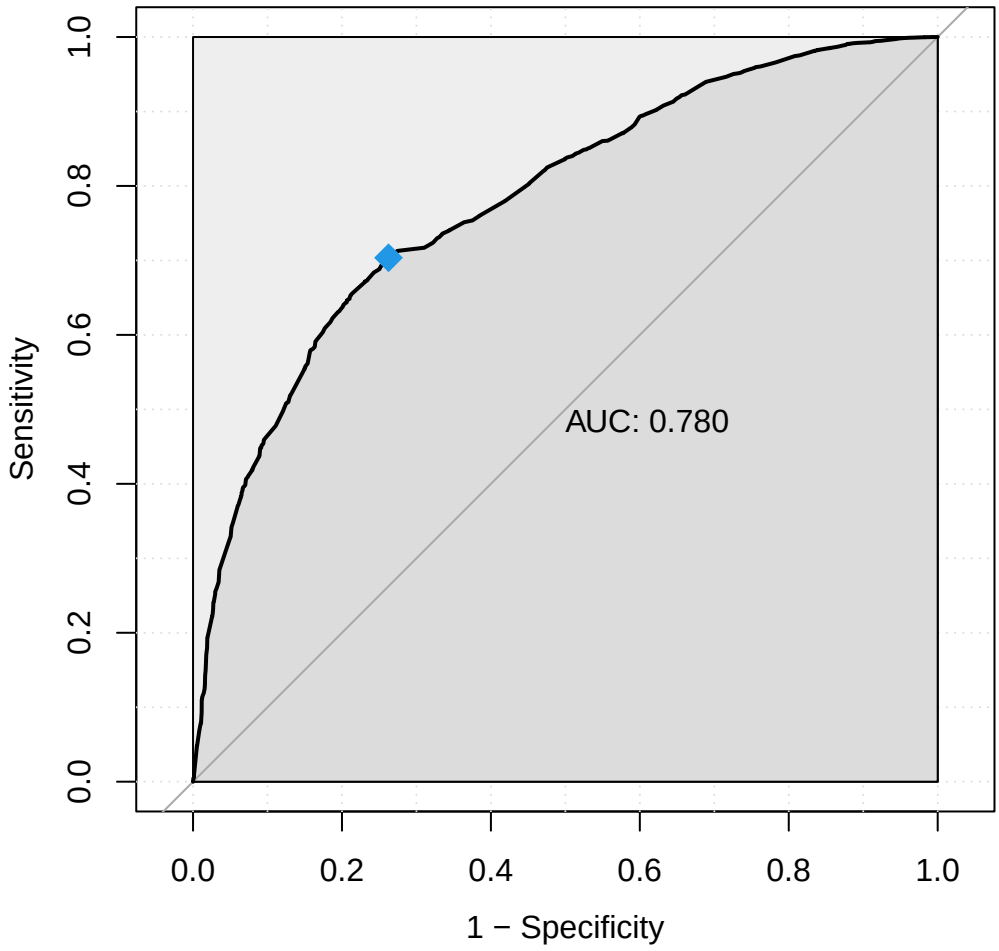

Figure 5.3.4: ROC curve of alteration of CL-adapt implemented in R.

The AUC obtained using this model is 0.78 which suggests that the classifier is superior to random guessing. Note that in figure 5.3.4, the coordinate marked in blue corresponds to the "optimal" threshold value of 0.491 which yields a sensitivity of 0.7035 and a specificity of 0.7374801. Thus if we were to decrese the probability threshold from 0.5 to 0.491, the model's sensitivity would increase while its specificity would decrease. We used the "optimal" threshold value to classify observations in the test set using this model. The accuracy returned from the latter approach was 0.7229454 which is a slight increase compared to the accuracy obtained when using 0.5 as the probability threshold.

5.3.2 FSSJ algorithm

Again, the bnclassify package in R served our purpose of fitting the FSSJ algorithm to our training data set. The algorithm was fitted using the function *fssj* with the following parameters:

- $\alpha = 0$. This indicates that MLE will be used to learn the conditional probabilities of the model.
- $k = 5$.
- $\epsilon = 0.01$

Recall that the use ϵ and k were defined in algorithm 5. Note that the properties of the smooth parameter α discussed in section 5.3.1 also hold for the *fssj* function. The confusion matrix obtained when fitting this model on the training set can be seen in figure 5.3.5. From the confusion matrix one can easily obtain the accuracy which is given by 0.7235, as well as the sensitivity and specificity which are 0.6512 and 0.7961 respectively. The F1 score of the model, when evaluated on the training set, is 0.7024.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	38902	17119
	Problem Gambler	9963	31957

Figure 5.3.5: Confusion matrix of FSSJ algorithm evaluated on the training data set

The accuracy of this model when evaluated on the test data set is 0.7239. From the confusion matrix, given in figure 5.3.6, we can easily derive the sensitivity and specificity of the model which are given by 0.6465 and 0.8012 respectively. Thus the model is relatively better at detecting non-problem gamblers than it is at detecting problem-gamblers. This is also evident from the confusion matrix since the number of falsely classified problem gamblers is considerably higher than the number of falsely classified non-problem gamblers. Since the precision of the model is 0.7646,

combining this with the sensitivity yields an F1 score of 0.701.

An interesting observation one can make when comparing the DAG of the model returned by the alteration of the CL-adapt algorithm given in figure 5.3.3 with that given in figure 5.3.7 is that out of the four variables that the FSSJ algorithm deemed useful for identifying problem gamblers, three of them were also included in the DAG returned by the former algorithm.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	13087	5767
	Problem Gambler	3247	10546

Figure 5.3.6: Confusion matrix of FSSJ algorithm evaluated on the test data set in R.

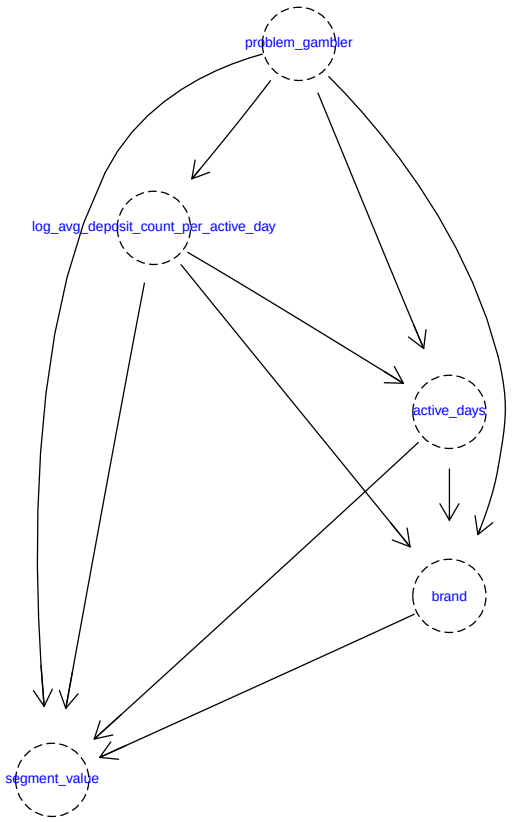

Figure 5.3.7: Bayesian network returned from the FSSJ algorithm implemented in R.

The DAG returned by the model can be seen in figure 5.3.7.Note that while the DAG does tell us which variables are influenced by the class variables, it does not yield any conditional independence statements when applying the Local Markov Property. This is because for each feature, the union of its set of non-descendants and its parent set is equal to the set of all variables.

Figure 5.3.8 below shows the ROC curve obtained for this algorithm. The "optimal" threshold value using the *closest to* $(0, 1)$ *criteria* is 0.457 with corresponding sensitivity and specificity equal to 0.691 and 0.753 respectively. Thus the "optimal threshold" increases the probability of correctly classifying a problem gambler but decreases the probability of correctly classifying a non problematic gambler. When using the "optimal" thereshold to classify observations in the test set using the model, the accuracy is given by 0.723068 which is a slight decrease compared to when the threshold was set to 0.5.

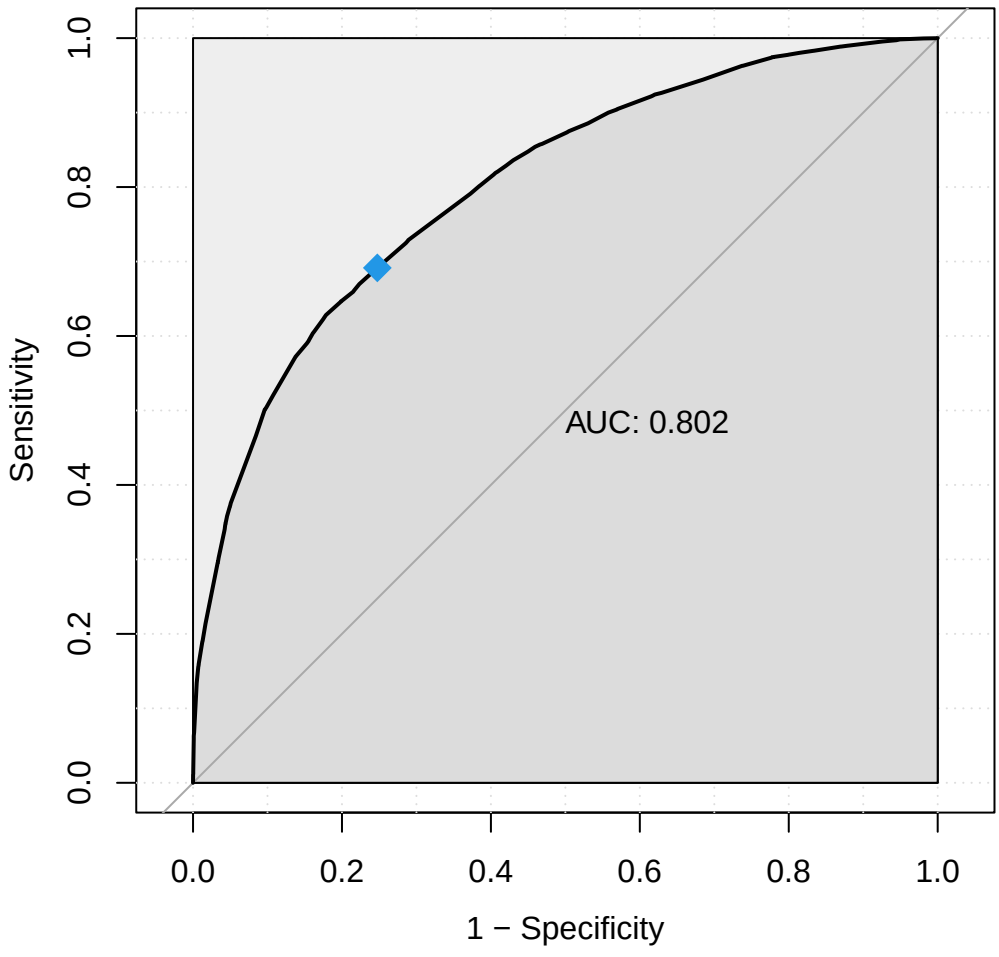

Figure 5.3.8: ROC curve of FSSJ algorithm

5.4 Tree-based methods for Identifying Problem Gamblers

In this section we shall go through the implementation of the random forest algorithm proposed by Breiman, as well as AdaBoost. These were both implemented in R using the **caret** package for the former and the **fastAdaboost** package for the latter.

5.4.1 Breiman's Random Forest algorithm

To implement algorithm 8 we used the *randomForest* function in R. We used the default values of B (number of trees) and m_{try} which are 500 and the square root of the number of features respectively. The confusion matrix returned by evaluating the model on the training data set can be seen in figure 5.4.1. The accuracy of this model is given by 0.951.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	46145	2084
	Problem Gambler	2720	46992

Figure 5.4.1: Confusion matrix of Breiman's Random Forest algorithm evaluated on the training set

When evaluating the model on the testing data set, we obtained the confusion matrix portrayed in figure 5.4.2 which yields an accuracy of 0.7883. Since the latter value is relatively different from the accuracy obtained when evaluating the model on the

training set, one may rightly be of the opinion that overfitting is present in the model. Moreover, the model yields a sensitivity of 0.7710, a specificity of 0.8056 and its precision is given by 0.7979. The $F1$ score of the model is 0.7842.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	13181	3729
	Problem Gambler	3181	12556

Figure 5.4.2: Confusion matrix of Breiman's Random Forest algorithm

The ROC curve of the model can be seen in figure 5.4.3 below. The AUC is given by 0.871 which suggests that the model has predictive ability superior to random guessing. The "optimal" threshold using the *closest to* $(0, 1)$ *criteria* is 0.475 with corresponding sensitivity and specificity given by 0.790482 and 0.7860286 respectively. When the model was tested on the test data set using the "optimal" threshold there was no change in accuracy up to 4 decimal places.

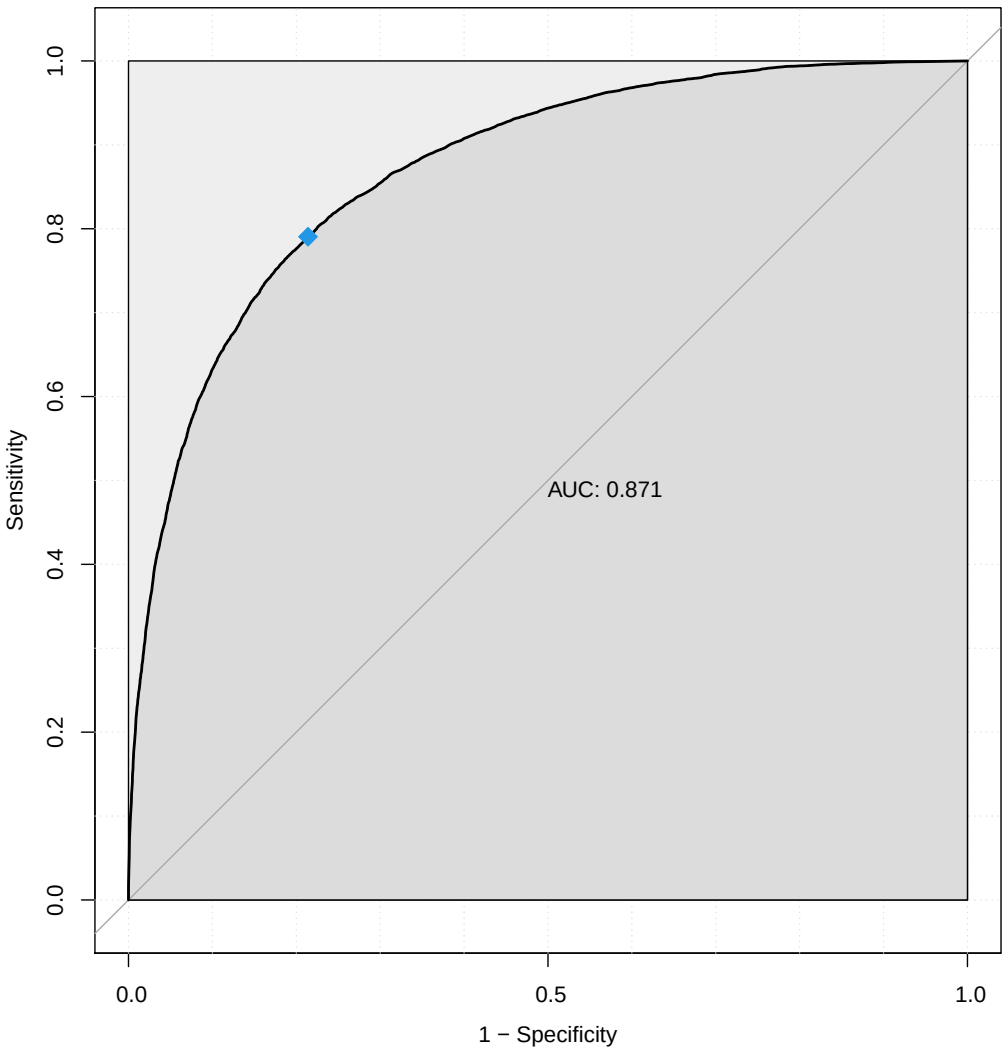

Figure 5.4.3: ROC curve of Breiman's Random Forest algorithm

5.4.2 AdaBoost

We used the **fastAdaboost** package in R to implement the AdaBoost algorithm. The number of weak learners (trees) to be used by the algorithm was set to the default value of 100. The confusion matrix obtained when evaluating this model on the training set can be seen in figure 5.4.4 below. This model obtained an accuracy of 0.979 coupled with a high $F1$ score of 0.9793. Moreover, the sensitivity and specificity obtained were 0.992 and 0.9655 respectively.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	47181	371
	Problem Gambler	1684	48705

Figure 5.4.4: Confusion matrix of AdaBoost algorithm on training data

From the confusion matrix shown in figure 5.4.5 we can calculate the accuracy which is given by 0.7671. This is relatively lower than the accuracy obtained on the training set, suggesting that overfitting is likely present in the AdaBoost model. The sensitivity and specificity of the model are given by 0.7918 and 0.7424 respectivey, which are both lower than the same measures evaluated on the training set. Moreover, the $F1$ score is given by 0.7723 which is also lower than it was when evaluated on the training set.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	12147	3390
	Problem Gambler	4215	12895

Figure 5.4.5: Confusion matrix of AdaBoost algorithm

The "optimal" threshold corresponding to the ROC curve in figure 5.4.6 is 0.4991 which yields a sensitivity of 0.7636 and a specificity of 0.7758. The accuracy of this model when using the "optimal" threshold is given by 0.7697185 which is a slight increase over the model which uses 0.5 as the probability threshold.

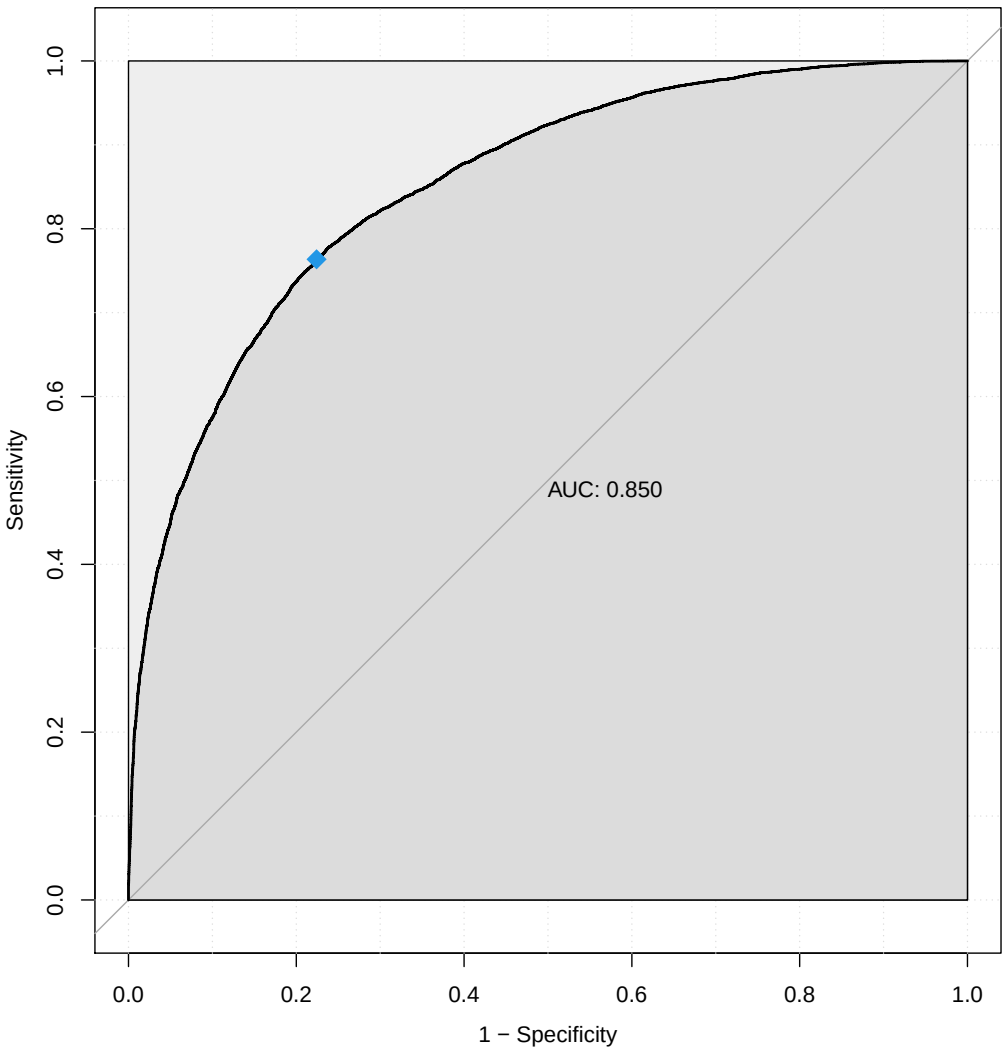

Figure 5.4.6: ROC curve of AdaBoost algorithm

5.5 Logistic Regression

In this section we outline the diagnostics of the logistic regression model obtained using the *glm* command in R. Prior to fitting the model, we performed a Chi-Square test of independence on each predictor variable and the response variable to check whether there is any association. It resulted that the p-values of all tests, apart from the one corresponding to the *segment_value* variable, were smaller than 0.05 hence implying that there is an association. However, as mentioned in a study by Bergh [6], the Chi-Square test is extremely sensitive to the sample size which in our case is 130588. This explains why most tests rejected the null hypothesis of independence.

In order to better understand the strength of these associations, the Cramér's V statistic was computed on the response variable and all predictor variables apart from "segment_value". We now explain how the aforementioned statistic is computed.

Let X_1 and X_2 be two categorical variables with k_1 and k_2 different categories respectively. Suppose $X_{11}, ..., X_{1n}$ are n independent observations of X_1 and $X_{21}, ..., X_{2n}$ of X_2. Let χ^2 be the Chi-Squared statistic computed using the frequency table of the n independent observations of X_1 and X_2. Then we have that

$$\text{Cramér's V} = \sqrt{\frac{\frac{\chi^2}{n}}{min(k_1 - 1, k_2 - 1)}}$$

This statistic gives a value between 0 and 1. The closer the value is to 1, the stronger the relationship between the two variables in consideration is.

The values obtained when computing the Cramér's V statistic using the predictor variables and the response variable can be seen in figure 5.2. We chose the variables with a Cramér's V value greater than 0.2, namely *active_days, depositing_days, log_avg_deposit_amount_per_active_day, log_avg_deposit_count_per_active_day* and *log_duration_hours_month* and checked for pairwise association between them using the Chi-Square test. As expected due to the large sample size, all p-values returned indicated that there is an association between each pair of said predictors. We therefore calculated the Cramér's V statistic to check the strength of each association. Table 5.3 contains the pairwise Cramér's V values between the five aforementioned variables.

Variable	Cramér's V
log_avg_deposit_count_per_active_day	0.3379
log_avg_deposit_amount_per_active_day	0.2888
active_days	0.2792
login_duration_hours_month	0.2228
depositing_days	0.221
avg_stake	0.1963
log_total_deposit_count	0.1961
brand	0.1931
log_avg_stake_per_active_day	0.1914
depositing_days_last_7_days	0.1838
country_code	0.1616
winsorized_game_win_margin	0.141
ggr_to_deposits_ratio	0.1359
season	0.1284
age_group	0.09348
login_duration_hours_week	0.08583
category_segment	0.07763
first_accepted_deposit_amount	0.05252
avg_session_hours_per_active_day	0.04027
log_total_money_bet_amount	0.03667
log_total_deposit_amount	0.02947

Table 5.2: Cramér's V measure of association between the response variable and the predictor variables which are associated with the response variable

Next, we considered two predictor variables having a Cramér's V value higher than 0.2 as being associated. In such a scenario, the variable having the smallest measure of association with the response variable was removed from the model.

	active_days	depositing_days	log_avg_deposit_amount_per_active_day	log_avg_deposit_count_per_active_day	log_duration_hours_month
active_days	1	0.8333	0.146	0.1223	0.6738
depositing_days	0.8333	1	0.1937	0.2101	0.6142
log_avg_deposit_amount_per_active_day	0.146	0.1937	1	0.4933	0.1731
log_avg_deposit_count_per_active_day	0.1223	0.2101	0.4933	1	0.1255
log_duration_hours_month	0.6738	0.6142	0.1731	0.1255	1

Table 5.3: Cramér's V measure between predictors

After the procedure explained above, the model was fitted with two variables, namely *active_days* and *log_avg_deposit_count_per_active_day*. As can be seen in table 5.5.1, the effects corresponding to both the predictor variables are significant.

Likelihood Ratio Tests

Effect	Model Fitting Criteria: -2 Log Likelihood of Reduced Model	Likelihood Ratio Tests: Chi-Square	df	Sig.
Intercept	683.794[a]	.000	0	.
active_days	11016.104	10332.310	2	.000
log_avg_deposit_count_per_active_day	14735.387	14051.592	2	.000

Figure 5.5.1: Likelihood Ratio Tests of logistic regression model

The confusion matrix attained when fitting the model to the training data set is portrayed in figure 5.5.2. From the latter one can evaluate the model's accuracy on the training data given by 0.6893.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	32803	14341
	Problem Gambler	16062	34735

Figure 5.5.2: Confusion matrix of Logistic Regression model on training data set

Figure 5.5.3 portrays the confusion matrix obtained on the testing data set. The accuracy attained is 0.6863 which is marginally lower than the accuracy of the model evaluated on the training data set, indicating that overfitting is likely not present. In figure 5.5.3, as well as in figure 5.5.2, the number of false positives is greater than the number of false negatives. The sensitivity and specificity of the model are given by 0.7078 and 0.6713 respectively. Other metrics will be reported in section 5.6.

		Actual	
		Not Problem Gambler	Problem Gambler
Predicted	Not Problem Gambler	10968	4874
	Problem Gambler	5366	11439

Figure 5.5.3: Confusion matrix of Logistic Regression model on testing data set

The ROC curve of the model can be seen in figure 5.4. The specificity and sensitivity corresponding to the "optimal" threshold value of 0.485 are given by 0.7315 and 0.7265 respectively, thus differing marginally compared to the values obtained when using 0.5 as the probability threshold.

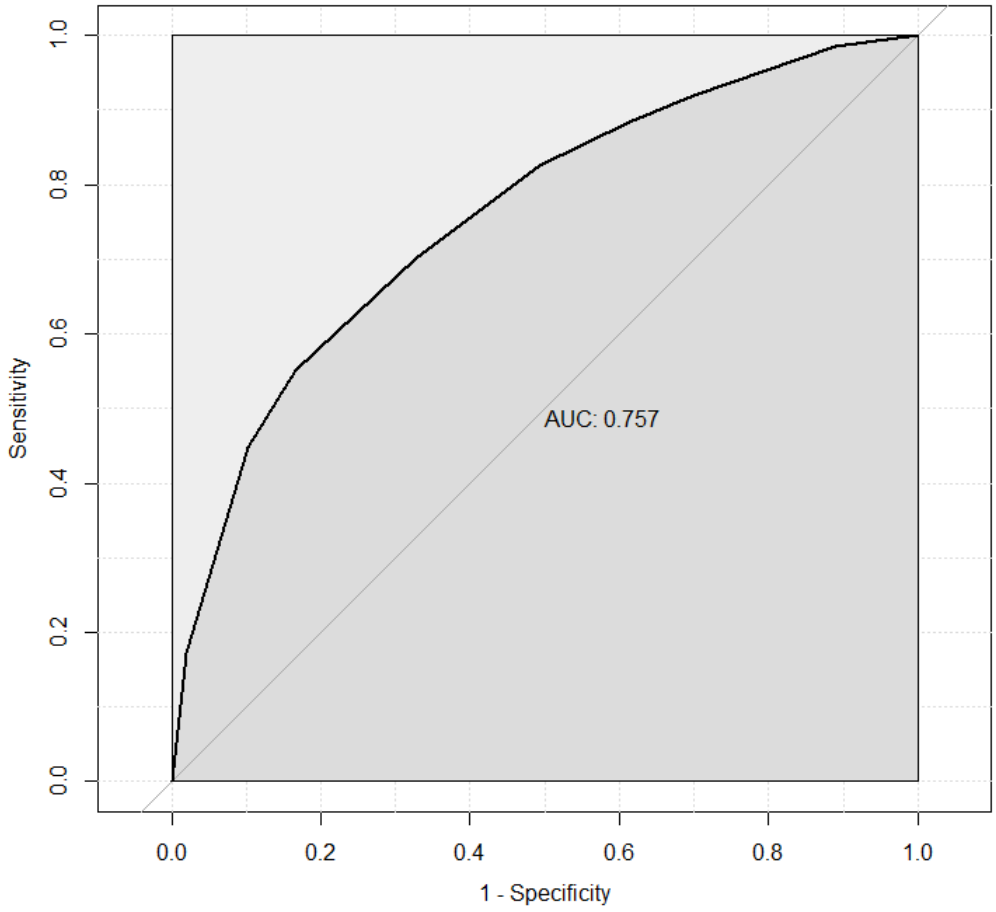

Table 5.4: ROC curve of Logistic Regression model

5.6 Comparison

This section will be dedicated to comparing the models we fitted in the previous sections, in addition to the logistic regression model which we fit in this section. We will summarize the goodness of fit measures and comment on how they differ across the different models. In order to be more concise, we will refer to the model returned by the modification of the CL-adapt algorithm as CL, that returned by the FSSJ algorithm as FSSJ, the AdaBoost model simply as AdaBoost, the model corresponding to Breiman's random forest algorithm as BRF and the logistic regression model as LogReg.

We commence the comparison by analysing the figures presented in table 5.5 below. Note that cells highlighted in grey correspond to the maximal goodness of fit measure amongst all models. The table suggests that on the whole, BRF is superior to all the other models we implemented. The accuracy of both the Bayesian techniques are immensely similar and relatively smaller than the accuracy obtained using BRF and AdaBoost. In terms of accuracy, the LogReg is inferior to the rest of the techniques. The LogReg model is inferior to the CL and FSSJ in all the metrics apart from sensitivity.

The AdaBoost and LogReg model have a sensitivity which is higher than their specificity which suggests that these models' ability to detect problem gamblers is superior to their ability of detecting non-problem gamblers. The contrary argument holds for the other 3 models.

		Measure				
Model	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
CL	0.72	0.6769	0.7464	0.732	0.7034	0.78
FSSJ	0.7239	0.6465	0.8012	0.7646	0.701	0.802
BRF	0.7883	0.771	0.8056	0.7979	0.7842	0.871
LogReg	0.6863	0.7012	0.6715	0.6807	0.6908	0.757
AdaBoost	0.7671	0.7918	0.7424	0.7537	0.7723	0.85

Table 5.5: Summary of Goodness of Fit Measures across all models

The ROC curves below suggest that all models have predictive ability superior to random guessing. In terms of AUC, the Bayesian techniques seem inferior to BRF and AdaBoost but superior to LogReg.

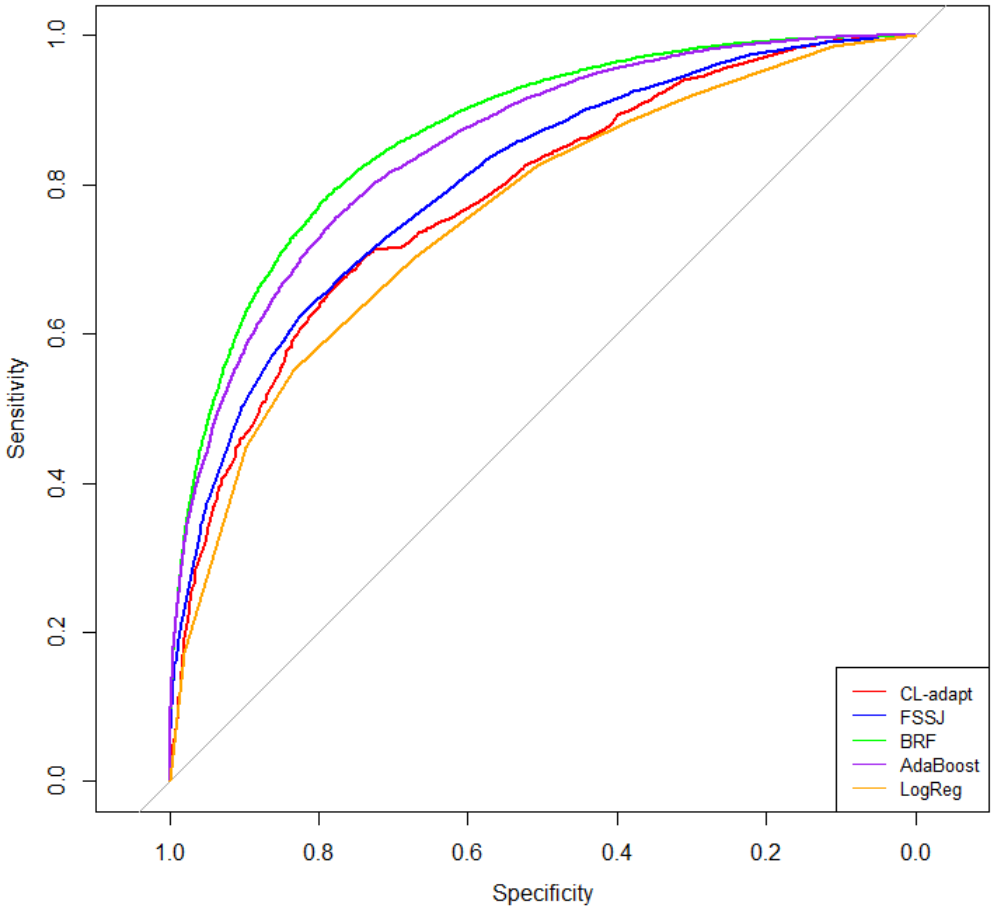

Figure 5.6.1: ROC curves of all models

Chapter 6

Conclusion

6.1 Review of Results

This chapter will conclude the study by highlighting the main results presented. It shall also review limitations of the study and propose opportunities of improvement for future research.

This study explored the efficacy of using Bayesian networks and random forests to predict problem gambling by using self-exclusion as a proxy. First, Bayesian networks were introduced as DAGs where directed edges indicated that a variable influences another. Consequently, we introduced Bayesian network classifiers which are BNs used to predict the class label of a response variable. We discussed how the structure of a BN classifier can be determined from a given data set using either the CL-adapt or the FSSJ algorithm. With regards to the CL-adapt algorithm, theorem 3.5.7 asserts that the likelihood of the data is maximized by said algorithm. However, we saw that the latter theoretical result comes at the expense of the algorithm placing strong assumptions on the graph structure, mainly that all features may have at most one parent apart from the class variable. On the other hand, the FSSJ algorithm is a heuristic which relaxes the strong assumptions placed by the CL-adapt algorithm.

The conditional probabilities, also referred to as the parameters of the BN, were in-

troduced and asymptotic results were presented. Most notably the MLE was proven to be a consistent estimator in theorem 3.6.8.

In addition to Bayesian networks,tree-based were introduced as another type of classifier. The algorithms of choice to build the structure of said methods were AdaBoost and Breiman's random forest algorithm.

The methods mentioned above, as well as Logistic regression, were applied on a data set provided by a local gaming company. In addition to these, we presented a modification of the CL-adapt algorithm which returned a model which was more parsimonious than that returned by the original CL-adapt algorithm at the slight expense of a marginally small difference in accuracy. Taking all goodness of fit diagnostics into account, the random forest classifiers performed better than the Bayesian techniques and the Logistic regression model. All models apart from the AdaBoost technique were superior in correctly identifying non-problem gamblers than problem-gamblers.

6.2 Further Considerations and Improvement

Although past studies, some of which mentioned in chapter 2 advocate for the use of self-exclusion as a proxy for problem gambling, this approach does have some inherent limitations as outlined by Griffiths & Auer [16]. The authors raise the point that there are various reasons for self-exclusion that have nothing to do with problem gambling, one of which being the player's annoyance with the low user experience quality on the operator's website. In an attempt to improve upon this dissertation, one could run an anonymous survey where the respondents would be individuals whose lives have been significantly socio-economically impacted by their gambling activities and individuals who gamble in a more social manner. Consequently, the assistance of psychologists, psychiatrists and other relevant professionals could be utilised in order to determine whether each individual should be labelled as a problem gambler or not. This procedure is both time consuming and expensive and

therefore was not considered for this dissertation.
A further limitation is derived from the lack of flexibility present in the package of choice for fitting Bayesian networks. As mentioned in sections 5.3.1 and 5.3.2, the functions of choice for fitting the Bayesian networks only support Bayesian parameter estimation in the special case when all the Dirichlet parameters are equal to a constant. To remedy this issue, one could attempt to build a package from scratch so as to be able to include Bayesian parameter estimation where all Dirichlet parameters are not necessarily equal. However this would be too time consuming and could not be done in the time frame allocated to me.

The models outlined in chapter 5 could be of practical use in the online gambling industry. If a particular user of a gambling site is predicted, by one or more of said models, to self exclude, an automated message could be sent to a customer representative of the betting operator who can decide whether or not said player should be contacted. Through a personalized email, the player could be asked to place monetary or time limits in order to reduce site activity and gamble in a more social than problematic manner.

This study could be further enhanced by implementing other machine learning techniques, such as neural networks which are a varied family of algorithms that aim to detect patterns in data through processes that mimic how the human brain functions. In particular,one may opt to consider Bayesian neural networks which combine the theory of Bayesian inference with that of neural networks. Another possible approach would be to reduce the dimensionality of the data set by using Principal Component Analysis (PCA) and fitting said neural networks to this transformed data set. Furthermore, it would be interesting to see other techniques including, but not limited to, Support Vector Machines (SVMs) and their Bayesian equivalent Relevant Vector Machines (RVM) applied in future work.

Appendix A

Definitions

Definition A.1. (Conditional independence) Let X, Y and Z be random variables. Then Y is said to be conditionally independent of X given Z, denoted by $Y \perp X|Z$, if $\forall(x, y, z)$,

$$P(y|x, z) = P(y|z).$$

Appendix B

Hyper-rectangles

Definition B.1. A hyper-rectangle in $\mathbb{R}^d$ where $d > 2$ is the generalization of a rectangle to a higher dimension d. A hyper-rectangle in $\mathbb{R}^2$ is simply a rectangle.

Example B.2. An example of a hyper-rectangle in $\mathbb{R}^3$ is the set

$$H = \{(x, y, z) \in \mathbb{R}^3 : -2 \leq x \leq 2, -1 \leq y \leq 1, -1 \leq z \leq 1\}.$$

Recall that in the proof of theorem 4.2.1, we said that for a fixed training set and a fixed vector of splits $\boldsymbol{\Theta}$, the set of all $\mathbf{x}$ such that $h(\mathbf{x}, \boldsymbol{\Theta}) = j$ is a union of hyper-rectangles. We shall demonstrate this using a simple example.

Example B.3. Suppose that

- X_1 is a categorical variable taking values in $\mathcal{X}_1 = \{1, 2, 3, 4\}$.
- X_2 is a categorical variable taking values in $\mathcal{X}_2 = \{1, 2, 3\}$.
- C is the binary classification variable taking values in $\{0, 1\}$.

Suppose we observed the sample given in table B.1.
Assume that, by using the CART algorithm, the decision tree in figure B.0.1 is returned.

X_1	X_2	C
1	1	0
1	2	0
1	3	1
2	1	0
2	2	0
2	3	1
3	1	1
3	2	1
3	3	1
4	1	0
4	2	1
4	3	1

Table B.1: Sample of observations of (X_1, X_2, C).

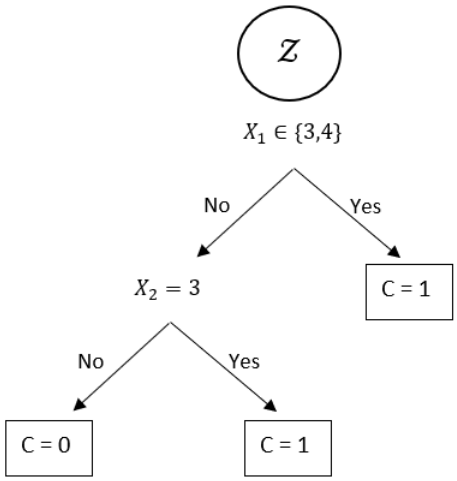

Figure B.0.1: Decision tree given data in table B.1

Consider the plot given in figure B.0.1. We can see that the classification region $\{\mathbf{x} \in \mathbb{R}^2 : h(\mathbf{x}, \Theta) = 1\}$ is given by the union of the hyper-rectangles A and B. On the other hand, the region $\{\mathbf{x} \in \mathbb{R}^2 : h(\mathbf{x}, \Theta) = 0\}$ is given by the hyper-rectangle C.

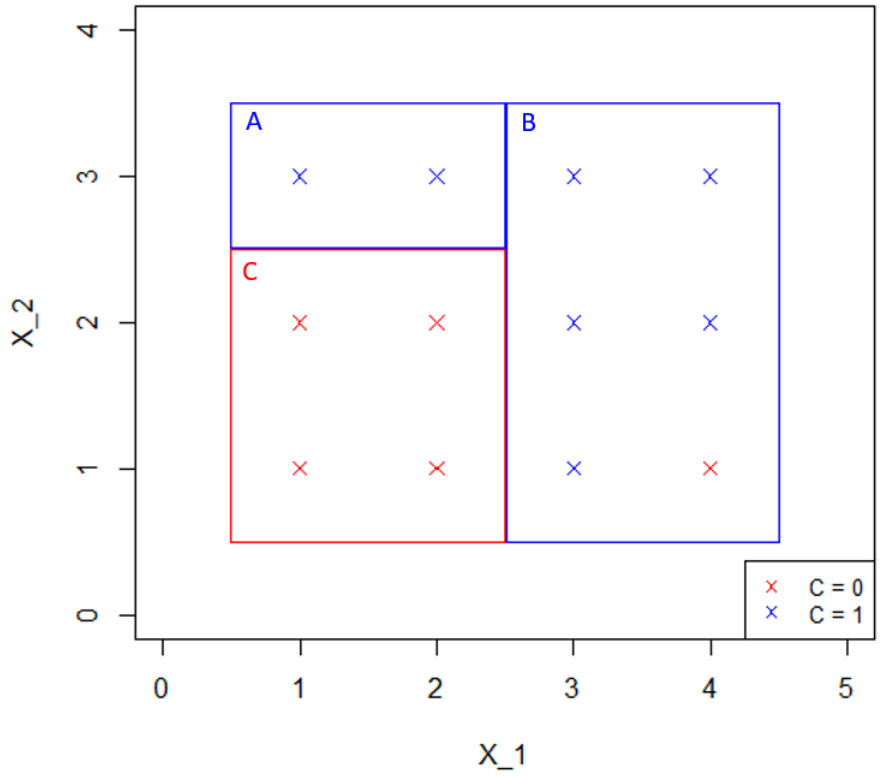

Figure B.0.2: Plot of observations together with hyper-rectangles which form the classification regions

Appendix C

Code

C.1 CL-adapt

```
1  library(bnclassify)
2  library(bnlearn)#for plot
3  library(Rcpp)
4  library(arules)
5  library(caret)#in addition to other things, this will be used for
       the confusion matrix
6  library(Rgraphviz)
7  library(adabag)#for adaboost
8  library(fastAdaboost)
9  library(PRROC)#for ROC curve
10 library(pROC)
11 library(randomForest)
12
13
14
15 df<-read.csv("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\Dataset\\
       Dataset_only_v2-csv.csv",header = TRUE)
16 length(unique(df$h_player_id))
```

```
17 #hist(df$log_avg_deposit_amount_per_active_day)
18 names(df)[names(df) == "lock_flag"] <- "problem_gambler"
19 names(df)[names(df) == "segment_value_on_exclusion_date"] <- "segment
       _value"
20
21 #removing all columns which are surely not useful.
22 df_sub<-df[ , !names(df) %in% c("h_player_id","lock_removed_date","
       lock_started","lock_ended","lock_planned_duration","is_definite_
       self_exclusion","week_after_payday","payday_week","age")]
23
24 #use the one below for the ROC
25 df_sub<-df[ , !names(df) %in% c("h_player_id","lock_removed_date","
       lock_started","lock_ended","lock_planned_duration","is_definite_
       self_exclusion","week_after_payday","payday_week")]
26
27 #converting char columns to numeric
28 df_sub[, names(df_sub) %in% c("avg_stake","first_accepted_deposit_
       amount","log_total_deposit_count","cancelled_withdrawals")]<-
       lapply(df_sub[, names(df_sub) %in% c("avg_stake","first_accepted_
       deposit_amount","log_total_deposit_count","cancelled_withdrawals"
       )],as.numeric)
29 help(discretizeDF)
30 #discretizing
31 disc_df<-discretizeDF(df_sub)
32
33 #removing cancelled_withdrawals because dicretizeDF only gave 1 bin
34 final_df<-disc_df[,!names(disc_df) %in% c("cancelled_withdrawals")]
35
36 final_df[]<- lapply( final_df, factor)
37
38 #removing_nulls
```

```
length(final_df$problem_gambler)#131432
#final_df_wout_nulls<-final_df[complete.cases(final_df),]

#use the below command instead of the above
final_df_wout_nulls<-na.omit(final_df)


#write.csv(final_df_wout_nulls,"C:\\Users\\chris\\OneDrive\\Desktop
    \\Thesis\\Thesis-RCode\\20220228-final-df.csv", row.names =
    FALSE)
final_df_wout_nulls<-read.csv("C:\\Users\\chris\\OneDrive\\Desktop\\
    Thesis\\Thesis-RCode\\20220228-final-df.csv",header = TRUE)

final_df_wout_nulls<-lapply(final_df_wout_nulls,factor)
final_df_wout_nulls<-as.data.frame(final_df_wout_nulls)
length(final_df_wout_nulls$problem_gambler)#130588

smp_size <- floor(0.75 * nrow(final_df_wout_nulls))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(final_df_wout_nulls)), size = smp_
    size)
#dividing into testing and training
train <- final_df_wout_nulls[train_ind, ]
test <- final_df_wout_nulls[-train_ind, ]

length(final_df_wout_nulls$problem_gambler)

#write.csv(train,"C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
    Thesis-RCode\\20220228-final-train-df.csv", row.names = FALSE)
```

```
#write.csv(test,"C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
    Thesis-RCode\\20220228-final-test-df.csv", row.names = FALSE)

str(train)
levels(final_df_wout_nulls$category_segment)

test_accuracy<-function(actual,predicted){
  len<-length(actual)
  cc<-vector()
  for (l in 1:len){
    if(predicted[l] == actual[l]){
      cc[l]<-1
    }else{
      cc[l]<-0
    }
  }
  acc<-sum(cc)/length(cc)
  return(acc)
}
#function to convert conditional probabilities into predictions
convert_cp_to_factor<-function(conditional_prob,threshold = 0.5){
  predicted<-vector()
  for (j in 1:nrow(conditional_prob)) {
    if(conditional_prob[j,1]<1-threshold){
      predicted[j]<-"TRUE"
    }else{
      predicted[j]<-"FALSE"
    }
  }
  return(as.factor(predicted))
}
```

```
95
96
97  #######Non-modified CL-adapt
98  CL_adapt_orig<-bnc('tan_cl', 'problem_gambler',dataset =train ,
        smooth = 0, dag_args = list(score = 'loglik'))
99  CL_adapt_orig<-bnc('tan_cl', 'problem_gambler',dataset =train ,
        smooth = 0)
100
101
102 plot(CL_adapt_orig)
103 orig_best_preds<-predict(CL_adapt_orig,test,prob=TRUE)
104 orig_best_predicted<-convert_cp_to_factor(orig_best_preds)
105 orig_best_accuracy<-test_accuracy(test$problem_gambler,orig_best_
        predicted)
106
107 ################################################################
108 #Where modification of CL-adapt algorithm actually starts
109
110
111
112
113 names<-names(final_df_wout_nulls)
114 len<-length(names)
115 #the algorithm depends on which starting variable you choose
116 starting_variable<-attributes[1]
117 attributes<-vector()
118 for (i in 1:len){
119   attributes[i]=names[i]
120 }
121 attributes<-attributes[attributes !="problem_gambler"]
122
```

```
123 models<-vector(mode = "list",length = length(attributes))#this will
       store the optimal model for each iteration. The difference
       between one iteration and
124 #another will be the starting variable.
125 accuracy_vec<-vector()
126 print(attributes[1])
127
128 for (j in 1:length(attributes)) {
129   starting_variable<-attributes[j]
130   epsilon<-0.01
131   diff<-1
132
133   attributes_not_being_used<-sample(attributes,length(attributes),
         replace = FALSE)#sample to introduce randomness
134   attributes_not_being_used<-attributes_not_being_used[attributes_not
         _being_used !=starting_variable]
135
136   chosen_attributes<-vector()
137   chosen_attributes[1]<-starting_variable
138
139   current_best_attribute<-starting_variable
140
141   #old_df<-final_df_wout_nulls[,names(final_df_wout_nulls) %in% c("
         problem_gambler",starting_variable,attributes_not_being_used
         [1])]
142   old_df<-train[,names(train) %in% c("problem_gambler",starting_
         variable,attributes_not_being_used[1])]
143   old_test_df<-test[,names(test) %in% c("problem_gambler",starting_
         variable,attributes_not_being_used[1])]
144   old_bn<-bnc('tan_cl', 'problem_gambler',dataset =old_df , smooth =
         0, dag_args = list(score = 'loglik'))
```

```
145  #old_accuracy<-cv(old_bn,old_test_df,k=10)
146  old_preds<-predict(old_bn,old_test_df,prob=TRUE)
147  old_predicted<-convert_cp_to_factor(old_preds)
148  old_accuracy<-test_accuracy(old_test_df$problem_gambler,old_
       predicted)
149  current_accuracy<-old_accuracy
150
151
152  while (diff>epsilon) {
153
154
155    for (attribute in attributes_not_being_used) {
156      #add attribute to list of currently chosen attributes
157      new_df<-train[,names(train) %in% c("problem_gambler",chosen_
           attributes,attribute)]
158      new_test_df<-test[,names(test) %in% c("problem_gambler",chosen_
           attributes,attribute)]
159      #fit tan_cl model on the new dataset
160      new_bn<-bnc('tan_cl', 'problem_gambler',dataset =new_df ,
           smooth = 0, dag_args = list(score = 'loglik'))
161      #new_accuracy<-cv(new_bn,new_test_df,k=10)
162      new_preds<-predict(new_bn,new_test_df,prob=TRUE)
163      new_predicted<-convert_cp_to_factor(new_preds)
164      new_accuracy<-test_accuracy(new_test_df$problem_gambler,new_
           predicted)
165
166      #decide whether BN with current attribute is better than BN
           with previous attribute
167      if (new_accuracy>current_accuracy){
168        current_accuracy<-new_accuracy
169        new_model<-new_bn
```

```
170         current_best_attribute<-attribute
171       }
172     }
173     #decide whether to add an attribute or not
174     diff<-current_accuracy-old_accuracy
175     if(diff>epsilon){
176       old_bn<-new_model
177       old_accuracy<-current_accuracy
178       attributes_not_being_used<-attributes_not_being_used[attributes
              _not_being_used!=current_best_attribute]
179       chosen_attributes<-append(chosen_attributes,current_best_
              attribute)
180     }else{
181       chosen_model<-old_bn
182       models[[j]]<-chosen_model
183       accuracy_vec[j]<-old_accuracy
184       #store old accuracy
185     }
186
187   }#of the while
188
189 }#of the for
190
191 ###### best cl adapt model####
192
193 #with new df
194
195 #saving models
196
197 saveRDS(models,file = "C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\20220308epsilon_1_percent_models.rds")
```

```
198 loaded_models<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\20220308epsilon_1_percent_models.rds")
199 #loaded_models<-load("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\models.rds")
200
201 saveRDS(accuracy_vec,file="C:\\Users\\chris\\OneDrive\\Desktop\\
       Thesis\\Thesis-RCode\\20220308epsilon_1_percent_accuracy_vec.rds"
       )
202 loaded_accuracy<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis
       \\Thesis-RCode\\20220308epsilon_1_percent_accuracy_vec.rds")
203
204
205
206 best_model<-loaded_models[[which.max(loaded_accuracy)]]
207 plot(best_model,fontsize=30,radius=700)
208
209 ##############################################
210 k<-1
211 for (model in loaded_models){
212   print(k)
213   print(model$.families)
214   plot(model,fontsize=40)
215   k<-k+1
216 }
217
218 max_index<-which.max(accuracy_vec)
219 accuracy_vec[max_index]
220 print(models[[1]]$.params)
221 #params_ij = P(row i | col J)
222
223
```

```
top6_accuracy<-head(ord <- order(accuracy_vec, decreasing = TRUE))
top6_models<-vector(mode="list",length = length(top6_accuracy))
for (j in 1:length(top6_accuracy)){
  top6_models[[j]]<-models[[top6_accuracy[j]]]
}

for (topmodel in top6_models){
  plot(topmodel,fontsize=40)
  print(topmodel$.families)
}

best_model<-loaded_models[[which.max(loaded_accuracy)]]
plot(best_model,fontsize=30,radius=700)

best_model$.params

plot(best_model,fontsize=40)
plot(loaded_models[which.max(loaded_accuracy)],fontsize=40)
chosen_model<-loaded_models[[which.max(loaded_accuracy)]]
plot(chosen_model,fontsize=32,radius=250)
graphviz.plot(best_model)
#log_avg_stake_per_active_day,problem_gambler,season,country_code,
    brand,active_days,segment_value_on_exclusion_date,log_total_
    deposit_amount,log_avg_deposit_count_per_active_day,

#par(mar=c(1,1,1,1)) to solve error figure margin too large


best_test_df<-test[,names(test) %in% c("problem_gambler","country_
    code","active_days","segment_value","log_avg_stake_per_active_day
    ","log_avg_deposit_count_per_active_day")]
```

```
251 best_train_df<-train[,names(test) %in% c("problem_gambler","country_
       code","active_days","segment_value","log_avg_stake_per_active_day
       ","log_avg_deposit_count_per_active_day")]
252 cv(chosen_model,best_train_df,dag=FALSE,k=10)
253
254 chosen_model$.params
255
256 best_preds<-predict(best_model,best_test_df,prob=TRUE)
257 best_predicted<-convert_cp_to_factor(best_preds)
258 best_accuracy<-test_accuracy(best_test_df$problem_gambler,best_
       predicted)
259
260 best_training_preds<-predict(best_model,best_train_df,prob=TRUE)
261 best_training_predicted<-convert_cp_to_factor(best_training_preds)
262 best_training_accuracy<-test_accuracy(best_train_df$problem_gambler,
       best_training_predicted)
263
264 head(best_preds)
265
266 head(best_predicted)
267
268 #test confusion matrix
269 confusionMatrix(data=best_predicted,reference = best_test_df$problem_
       gambler,positive = "TRUE")
270
271 #train confusion matrix
272 confusionMatrix(data=best_training_predicted,reference = best_train_
       df$problem_gambler,positive = "TRUE")
```

C.2 FSSJ algorithm

```
1
2  ##FSSJ algorithm
3  library(bnclassify)
4  library(bnlearn)#for plot
5  library(Rcpp)
6  library(arules)
7  library(caret)#in addition to other things, this will be used for
       the confusion matrix
8  library(Rgraphviz)
9  df<-read.csv("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\Dataset\\
       Dataset_only_v2-csv.csv",header = TRUE)
10 names(df)[names(df) == "lock_flag"] <- "problem_gambler"
11 View(df)
12 names(df)
13 #removing all columns which are surely not useful
14 df_sub<-df[ , !names(df) %in% c("h_player_id","lock_removed_date","
       lock_started","lock_ended","lock_planned_duration","is_definite_
       self_exclusion")]
15 #converting char columns to numeric
16 df_sub[, names(df_sub) %in% c("avg_stake","first_accepted_deposit_
       amount","log_total_deposit_count","cancelled_withdrawals")]<-
       lapply(df_sub[, names(df_sub) %in% c("avg_stake","first_accepted_
       deposit_amount","log_total_deposit_count","cancelled_withdrawals"
       )],as.numeric)
17
18 #discretizing
19 disc_df<-discretizeDF(df_sub)
20 #removing cancelled_withdrawals because dicretizeDF only gave 1 bin
21 final_df<-disc_df[,!names(disc_df) %in% c("cancelled_withdrawals")]
22 final_df[] <- lapply( final_df, factor)
23
```

```
24 #removing_nulls
25 length(final_df$problem_gambler)#131432
26 final_df_wout_nulls<-final_df[complete.cases(final_df),]
27 length(final_df_wout_nulls$problem_gambler)#130588
28
29 smp_size <- floor(0.75 * nrow(final_df_wout_nulls))
30
31 ## set the seed to make your partition reproducible
32 set.seed(123)
33 train_ind <- sample(seq_len(nrow(final_df_wout_nulls)), size = smp_
      size)
34 #dividing into testing and training
35 train <- final_df_wout_nulls[train_ind, ]
36 test <- final_df_wout_nulls[-train_ind, ]
37
38
39 fssj_model<-bnc("fssj","problem_gambler",train,smooth=0,dag_args =
      list(k=5,epsilon=0.01))
40 fssj_pred_probs<-predict(fssj_model,test,prob=TRUE)
41 fssj_preds<-predict(fssj_model,test,prob=FALSE)
42 head(fssj_preds)
43
44 fssj_model$.params
45
46 test_accuracy(test$problem_gambler,fssj_preds)
47
48 plot(fssj_model,fontsize=40)
49
50 best_preds<-predict(best_model,best_test_df,prob=TRUE)
51 best_predicted<-convert_cp_to_factor(best_preds)
52 best_accuracy<-test_accuracy(best_test_df$problem_gambler,best_
```

```
    predicted)
53
54 training_fssj_preds<-predict(fssj_model,train,prob=TRUE)
55 training_fssj_predicted<-convert_cp_to_factor(training_fssj_preds)
56 training_fssj_accuracy<-test_accuracy(train$problem_gambler,training_
    fssj_predicted)
57
58 #test confusion matrix
59 confusionMatrix(data=fssj_preds,reference = test$problem_gambler,
    positive = "TRUE")
60
61 #training confusion matrix
62 confusionMatrix(data=training_fssj_predicted,reference = train$
    problem_gambler,positive = "TRUE")
```

C.3 BRF algorithm

```
1  library(randomForest)
2
3  random_forest<-randomForest(problem_gambler~.,data=train,ntree=500)
4  saveRDS(random_forest,file = "C:\\Users\\chris\\OneDrive\\Desktop\\
    Thesis\\Thesis-RCode\\20220129-random-forest.rds")
5  random_forest<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
    Thesis-RCode\\20220129-random-forest.rds")
6
7  rf_preds<-predict(random_forest,test,type="prob")
8  rf_predicted_values<-predict(random_forest,test)
9  head(rf_predicted_values)
10 head(rf_preds)
11
```

```
12 ##for training data
13 training_rf_preds<-predict(random_forest,train,type="prob")
14 training_rf_predicted_values<-predict(random_forest,train)
15
16 #training confusion matrix
17 confusionMatrix(data=training_rf_predicted_values,reference = train$
       problem_gambler,positive = "TRUE")
18
19 #testing conf matrix for random forest
20
21 confusionMatrix(data=rf_predicted_values,reference = best_test_df$
       problem_gambler,positive = "TRUE")
```

C.4 AdaBoost

```
1  library(fastAdaboost)
2  ada_pred<-predict(adamodel_fast_loaded,test,prob=TRUE)
3  ada_pred_training<-predict(adamodel_fast_loaded,train)
4  head(ada_pred_training$class)
5  #conf matrix for training data
6  confusionMatrix(data=ada_pred_training$class,reference = train$
       problem_gambler,positive = "TRUE")
7
8  #conf matrix for testing data set
9  ada_pred_testing<-predict(adamodel_fast_loaded,test)
10 confusionMatrix(data=ada_pred_testing$class,reference=test$problem_
       gambler,positive="TRUE")
11 head(ada_pred)
```

C.5 Logistic Regression

```
1  library(haven)
2  library(rcompanion)
3  library(dplyr)
4
5
6  names<-names(final_df_wout_nulls)
7  len<-length(names)
8  attributes<-vector()
9  removed_attributes<-vector()
10
11
12 for (i in 1:len){
13   attributes[i]=names[i]
14 }
15 attributes<-attributes[attributes !="problem_gambler"]
16
17 C<-final_df_wout_nulls$problem_gambler
18
19 #Step1: pairwise chisquare between respone and all predictors
20
21 chisq_matrix<-matrix(0,nrow = length(attributes),2)
22
23 for (i in 1:length(attributes)){
24   var1<-final_df_wout_nulls[,names %in% c(attributes[i])]
25   p_value_chisq<-chisq.test(var1,C)$p.value
26   chisq_matrix[i,2]<-p_value_chisq
27   chisq_matrix[i,1]<-attributes[i]
28   }
29
```

```
chisq_df<-as.data.frame(chisq_matrix)

write.excel <- function(x,row.names=FALSE,col.names=TRUE,...) {
  write.table(x,"clipboard",sep="\t",row.names=row.names,col.names=
      col.names,...)
}

write.excel(chisq_df)

#Step 2: CramerV with response and predictors

cramer_response_matrix<-matrix(0,nrow = length(attributes),2)


for (i in 1:length(attributes)){
  var1<-final_df_wout_nulls[,names %in% c(attributes[i])]
  cramer<-cramerV(table(var1,C))
  cramer_response_matrix[i,2]<-cramer
  cramer_response_matrix[i,1]<-attributes[i]

}
cramer_response_matrix
colnames(cramer_response_matrix)<-c("attribute","cramer")
cramer_response_df<-as.data.frame(cramer_response_matrix)

cramer_subset<-cramer_response_df[cramer_response_df$cramer >0.25]

cramer_subset<-cramer_response_df %>%
  filter(cramer > 0.2)

write.excel(cramer_response_df)
```

```
attributes_cramer_response<-cramer_subset$attribute

#Step 3: Pairwise Chisq between variables in attributes_cramer_
    response

chisq_pairwise<-matrix(0,nrow = length(attributes_cramer_response),
    ncol=length(attributes_cramer_response))
rownames(chisq_pairwise)<-attributes_cramer_response

for (i in 1:length(attributes_cramer_response)){
  for (j in 1:length(attributes_cramer_response)) {

    if(i!=j){
      var1<-final_df_wout_nulls[,names %in% c(attributes_cramer_
          response[i])]
      var2<-final_df_wout_nulls[,names %in% c(attributes_cramer_
          response[j])]
      chisq_pairwise[i,j]<-chisq.test(var1,var2)$p.value
    }
  }
}

#Step 4: Pairwise CramerV between variables in attributes_cramer_
    response


cramer_pairwise<-matrix(0,nrow = length(attributes_cramer_response),
    ncol=length(attributes_cramer_response))
rownames(cramer_pairwise)<-attributes_cramer_response


```

```
85 for (i in 1:length(attributes_cramer_response)){
86   for (j in 1:length(attributes_cramer_response)) {
87
88       var1<-final_df_wout_nulls[,names %in% c(attributes_cramer_
             response[i])]
89       var2<-final_df_wout_nulls[,names %in% c(attributes_cramer_
             response[j])]
90       cramer_pairwise[i,j]<-cramerV(table(var1,var2))
91
92   }
93 }
94
95 cramer_pairwise
96
97
98 cramer_pairwise_df<-as.data.frame(cramer_pairwise)
99 write.excel(cramer_pairwise_df)
100
101
102 #Step 5: Remove associated variables
103
104
105 removed_attributes<-vector()
106
107 for (i in 1:length(attributes_cramer_response)){
108   for (j in 1:length(attributes_cramer_response)) {
109
110     if(i!=j){
111       var1<-final_df_wout_nulls[,names %in% c(attributes_cramer_
             response[i])]
112       var2<-final_df_wout_nulls[,names %in% c(attributes_cramer_
```

```
            response[j])]
113       cramer_pairwise[i,j]<-cramerV(table(var1,var2))
114
115       if(cramer_pairwise[i,j] > 0.2){
116         v1cv<-cramerV(table(var1,C))
117         v2cv<-cramerV(table(var2,C))
118
119         if (v1cv>v2cv){
120           removed_attributes<-append(removed_attributes,attributes_
                cramer_response[j])#remove var2
121         }else{
122           removed_attributes<-append(removed_attributes,attributes_
                cramer_response[i])#remove var1
123         }
124       }
125     }
126   }
127 }
128
129 unique(removed_attributes)
130
131
132 chosen_attributes<-attributes_cramer_response[!(attributes_cramer_
       response %in% unique(removed_attributes))]
133
134 log_train<-train[,names(train) %in% append(chosen_attributes,"problem
       _gambler")]
135 log_test<-test[,names(test) %in% append(chosen_attributes,"problem_
       gambler")]
136
137 log_reg_model_w_mc_diag <- glm(problem_gambler ~.,family=binomial(
```

```
    link='logit'),data=log_train)
138
139 summary(log_reg_model_w_mc_diag)
140
141 #getting training confusion matrix
142 train_log_reg_preds<-predict(log_reg_model_w_mc_diag,log_train,type="
    response")
143 train_log_reg_predicted_class_mc<-as.factor(ifelse(train_log_reg_
    preds>0.5,"TRUE","FALSE"))
144 confusionMatrix(data=train_log_reg_predicted_class_mc,reference = log
    _train$problem_gambler,positive="TRUE")
145
146 str(train_log_reg_predicted_class_mc)
147
148 head(train_log_reg_predicted_class_mc)
149
150 ##using only variable which was returned after running
    multicollinearity diagnostics
151
152 log_reg_preds_mc<-predict(log_reg_model_w_mc_diag,log_test,type="
    response")
153 log_reg_predicted_class_mc<-as.factor(ifelse(log_reg_preds_mc>0.5,"
    TRUE","FALSE"))
154 confusionMatrix(data=log_reg_predicted_class_mc,reference = test$
    problem_gambler,positive="TRUE")
```

C.6 Comparison

```
1
2 #Comparing ROC curves
```

```
3
4  #cl_adapt predictions
5  loaded_models<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\20220129epsilon_1_percent_models.rds")
6  loaded_accuracy<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis
       \\Thesis-RCode\\20220129epsilon_1_percent_accuracy_vec.rds")
7  best_model<-loaded_models[[which.max(loaded_accuracy)]]
8  plot(best_model)
9
10 best_test_df<-test[,names(test) %in% c("problem_gambler","country_
       code","active_days","segment_value","log_avg_stake_per_active_day
       ","log_avg_deposit_count_per_active_day")]
11 best_train_df<-train[,names(test) %in% c("problem_gambler","country_
       code","active_days","segment_value","log_avg_stake_per_active_day
       ","log_avg_deposit_count_per_active_day")]
12 best_preds<-predict(best_model,best_test_df,prob=TRUE)
13
14 ROC_CL_adapt <- roc(best_test_df$problem_gambler,best_preds[,2],
15                     smoothed = FALSE,
16                     # arguments for ci
17                     ci=FALSE, ci.alpha=0.9, stratified=FALSE,
18                     # arguments for plot
19                     plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE,
                           grid=TRUE,
20                     print.auc=TRUE, show.thres=TRUE,legacy.axes = TRUE)
21
22 ##fssj
23 fssj_model<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\20220130-fssj_model.rds")
24
25 fssj_pred_probs<-predict(fssj_model,test,prob=TRUE)
```

```
26
27
28 ROC_FSSJ <- roc(test$problem_gambler,fssj_pred_probs[,2],
29                 smoothed = FALSE,
30                 # arguments for ci
31                 ci=FALSE, ci.alpha=0.9, stratified=FALSE,
32                 # arguments for plot
33                 plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE,
                       grid=TRUE,
34                 print.auc=TRUE, show.thres=TRUE,legacy.axes = TRUE)
35
36
37 ##random forest
38 random_forest<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
       Thesis-RCode\\20220129-random-forest.rds")
39
40 rf_preds<-predict(random_forest,test,type="prob")
41 head(rf_preds)
42 roc_random_forest<-roc(test$problem_gambler,rf_preds[,2],
43                        smoothed = FALSE,
44                        # arguments for ci
45                        ci=FALSE, ci.alpha=0.9, stratified=FALSE,
46                        # arguments for plot
47                        plot=TRUE, auc.polygon=TRUE, max.auc.polygon=
                              TRUE, grid=TRUE,
48                        print.auc=TRUE, show.thres=TRUE,legacy.axes =
                              TRUE)
49
50 ##adaboost
51
52 adamodel_fast_loaded<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\
```

```
    Thesis\\Thesis-RCode\\20220202-AdaBoost.rds")
ada_pred<-predict(adamodel_fast_loaded,test,type="prob")
head(ada_pred)

roc_ada_boost<-roc(test$problem_gambler,ada_pred$prob[,2],
                   smoothed = FALSE,
                   # arguments for ci
                   ci=FALSE, ci.alpha=0.9, stratified=FALSE,
                   # arguments for plot
                   plot=TRUE, auc.polygon=TRUE, max.auc.polygon=
                       TRUE, grid=TRUE,
                   print.auc=TRUE, show.thres=TRUE,legacy.axes =
                       TRUE)

##Logistic regression
#run code in Log-reg-testing-assumptions

log_reg_df<-read.csv("C:\\Users\\chris\\OneDrive\\Desktop\\Thesis\\
    SPSS\\20220323-log-reg-test-preds.csv",header = TRUE)
head(log_reg_df)


#log_reg_model<-readRDS("C:\\Users\\chris\\OneDrive\\Desktop\\
    Thesis\\Thesis-RCode\\20220214-Log-Reg.rds")
#log_reg_preds<-predict(log_reg_model,test,type="response")
#log_reg_predicted_class<-as.factor(ifelse(log_reg_preds>0.5,"TRUE
    ","FALSE"))


log_reg_preds_mc<-predict(log_reg_model_w_mc_diag,log_test,type="
    response")
```

```
77 log_reg_predicted_class_mc<-as.factor(ifelse(log_reg_preds_mc>0.5,"
       TRUE","FALSE"))
78
79
80 #the below has been updated to use predictions obtained using SPSS.
        These are found in the data frame log_reg_df
81 View(log_reg_predicted_class)
82
83 roc_log_reg<-roc(test$problem_gambler,log_reg_df$Prob_true,
84                  smoothed = FALSE,
85                  # arguments for ci
86                  ci=FALSE, ci.alpha=0.9, stratified=FALSE,
87                  # arguments for plot
88                  plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE,
                        grid=TRUE,
89                  print.auc=TRUE, show.thres=TRUE,legacy.axes = TRUE)
90
91
92 log_opt_threshold<-coords(roc_log_reg, "best", ret = "threshold",best
       .method = "closest.topleft")#0.4942058
93 log_x_best<-coords(roc_log_reg, "best", ret = "specificity",best.
       method = "closest.topleft")
94 log_y_best<-coords(roc_log_reg, "best", ret = "sensitivity",best.
       method = "closest.topleft")
95 points(log_x_best, log_y_best, type = "p",pch=18,col=4,cex=2)
96
97 log_reg_predicted_class_opt<-as.factor(ifelse(log_reg_preds_mc>log_
       opt_threshold$threshold,"TRUE","FALSE"))
98
99 confusionMatrix(data=log_reg_predicted_class_opt,reference = test$
       problem_gambler,positive = "TRUE")
```

```
100
101 ####all ROC curves
102
103 plot(ROC_CL_adapt,col="red")
104 plot(ROC_FSSJ,col="blue",add=TRUE)
105 plot(roc_random_forest,col="green",add=TRUE)
106 plot(roc_ada_boost,col="purple",add=TRUE)
107 plot(roc_log_reg,col="orange",add=TRUE)
108
109 legend("bottomright", legend=c("CL-adapt", "FSSJ","BRF","AdaBoost","
       LogReg"),
110        col=c("red", "blue","green","purple","orange"),lty=1, cex=0.8)
```

Bibliography

[1] Nicola Adami et al. "Markers of unsustainable gambling for early detection of at-risk online gamblers". In: *International Gambling Studies* 13 (Aug. 2013), pp. 188–204. DOI: 10.1080/14459795.2012.754919 (cit. on p. 5).

[2] Rupesh Agrahari et al. "Applications of Bayesian network models in predicting types of hematological malignancies." In: *Sci Rep 8* (2018). DOI: https://doi.org/10.1038/s41598-018-24758-5 (cit. on p. 7).

[3] Syed Akhter. "Using machine learning to predict potential online gambling addicts." English. Master's thesis. Aalto University. School of Science, 2018, p. 45. URL: http://urn.fi/URN:NBN:fi:aalto-201810175429 (cit. on p. 4).

[4] Michael Auer and Mark D Griffiths. "Predicting Limit-Setting Behavior of Gamblers Using Machine Learning Algorithms: A Real-World Study of Norwegian Gamblers Using Account Data". In: *International Journal of Mental Health and Addiction* (2019). DOI: https://doi.org/10.1007/s11469-019-00166-2 (cit. on p. 6).

[5] Daniel Berend and Aryeh Kontorovich. "On the Convergence of the Empirical Distribution". In: (May 2012) (cit. on p. 44).

[6] Daniel Bergh. "Sample Size and Chi-Squared Test of Fit—A Comparison Between a Random Sample Approach and a Chi-Square Value Adjustment Method Using Swedish Adolescent Data". In: (2015). Ed. by Quan Zhang and Hong Yang, pp. 197–211 (cit. on p. 81).

[7] Måns Bermell. "Identification of Problem Gambling via Recurrent Neural Networks: Predicting self-exclusion due to problem gambling within the remote gambling sector by means of recurrent neural networks". In: (2019). URL:

https://www.semanticscholar.org/paper/Identification-of-Problem-Gambling-via-Recurrent-to-Bermell/d61432ef6870cde1765b1b7b4c4df72ce92ed59
(cit. on p. 6).

[8] Leo Breiman et al. *Classification and Regression Trees*. Chapman and Hall, 1984, pp. 26–109. DOI: https://doi.org/10.1201/9781315139470 (cit. on pp. 51, 53, 57).

[9] Kurt Dylan Buttigieg, Mark Anthony Caruana, and David Suda. "Identifying Problematic Gamblers using Multiclass and Two-stage Binary Neural Network Approaches". eng. In: (2021) (cit. on p. 6).

[10] David Maxwell Chickering. "Learning Bayesian Networks is NP-Complete". In: *Learning from Data: Artificial Intelligence and Statistics V*. Ed. by Doug Fisher and Hans-J. Lenz. New York, NY: Springer New York, 1996, pp. 121–130. ISBN: 978-1-4612-2404-4. DOI: 10.1007/978-1-4612-2404-4_12. URL: https://doi.org/10.1007/978-1-4612-2404-4_12 (cit. on p. 30).

[11] T.M Cover and J.A Thomas. *Elements of Information Theory*. John Wiley Sons, 1991, pp. 14–25 (cit. on p. 32).

[12] Riza Demirer, Ronald R Mau, and Catherine Shenoy. "Bayesian networks: a decision tool to improve portfolio risk analysis". In: *Journal of applied finance* 16.2 (2006), p. 106. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.1084&rep=rep1&type=pdf (cit. on p. 8).

[13] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X (cit. on pp. 51, 61).

[14] Olivier Gevaert et al. "Predicting the prognosis of breast cancer by integrating clinical and microarray data with Bayesian networks". In: *Bioinformatics* 22.14 (July 2006), e184–e190. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl230. eprint: https://academic.oup.com/bioinformatics/article-

pdf/22/14/e184/615668/btl230.pdf. URL: https://doi.org/10.1093/bioinformatics/btl230 (cit. on p. 7).

[15] Mark Griffiths et al. "Sociodemographic Correlates of Internet Gambling:Findings from the 2007 British Gambling Prevalence Survey". In: *CyberPsychology Behaviour* 12.2 (2009), pp. 199–202 (cit. on p. 1).

[16] Michael Auer Mark Griffiths. "Should voluntary "self exclusion" by gamblers be used as a proxy measure for problem gambling?" In: *MOJ Addict Med Ther* 2 (2 2016), pp. 31–33. DOI: 10.15406/mojamt.2016.02.00019 (cit. on p. 89).

[17] Pedro Hubert Mark Griffiths. "A Comparison of Online Versus Offline Gambling Harm in Portuguese Pathological Gamblers: An Empirical Study". In: *International Journal of Mental Health Addiction* 16 (2018), pp. 1219–1237 (cit. on p. 2).

[18] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009, pp. 698–704 (cit. on pp. 9, 11, 43).

[19] Timo Koski and John M. Noble. *Bayesian Networks: an introduction*. John Wiley and Sons,Ltd, 2009 (cit. on pp. 9, 12, 14).

[20] M.Minsky. "Steps toward artificial intelligence". In: *Transactions on Institute of Radio Engineers* 49 (1961), pp. 8–30 (cit. on p. 17).

[21] M. M. Mansour, Mohamed A. A. Wahab, and Wael M. Soliman. "Bayesian Networks for Fault Diagnosis of a Large Power Station and its Transmission Lines, Electric Power Components and Systems". In: (2012). DOI: 10.1080/15325008.2012.666615 (cit. on p. 7).

[22] Abby McCormack and Mark Griffiths. "Motivating and Inhibiting Factors in Online Gambling Behaviour: A Grounded Theory Study". In: *International Journal of Mental Health Addiction* 10 (2012), pp. 39–53 (cit. on p. 2).

[23] Bojan Mihaljevic, Concha Bielza, and Pedro Larranaga. "bnclassify: Learning Bayesian Network Classifiers". In: *The R Journal* 10 (Jan. 2019), p. 455. DOI: 10.32614/RJ-2018-073 (cit. on p. 68).

[24] N.Friedman, D.Geiger, and M.Goldszmidt. "Bayesian network classifiers". In: *Machine Learning* 29 (1997), pp. 131–163 (cit. on pp. 19, 20, 30).

[25] M. Pazzani. "Constructive induction of Cartesian product attributes". In: *Proceedings of the Information, Statistics and Induction in Science Conference* (1996), pp. 66–77 (cit. on p. 19).

[26] Christian Percy et al. "Predicting online gambling self-exclusion: An analysis of the performance of supervised machine learning models." In: *International Gambling Studies* 16 (2 2016), pp. 193–210. DOI: 10.1080/14459795.2016.1151913 (cit. on p. 5).

[27] Neil J. Perkins and Enrique F.Schisterman. "The Inconsistency of "Optimal" Cutpoints Obtained using Two Criteria based on the Receiver Operating Characteristic Curve". In: *American Journal of Epidemiology* 163(7) (2006), pp. 670–675. DOI: https://doi.org/10.1093/aje/kwj063 (cit. on p. 67).

[28] N. M Petry. "Internet gambling: An emerging concern in family practice medicine?" In: *Family Practice* 23 (2006), pp. 421–426 (cit. on p. 1).

[29] Kahlil S. Philander. "Identifying high-risk online gamblers: a comparison of data mining procedures". In: *International Gambling Studies* 14.1 (2014), pp. 53–63. DOI: 10.1080/14459795.2013.841721. eprint: https://doi.org/10.1080/14459795.2013.841721. URL: https://doi.org/10.1080/14459795.2013.841721 (cit. on pp. 5, 53).

[30] Lena Quilty, R Michael Bagby, and Murati D Avila. "Identifying indicators of harmful and problem gambling in a Canadian sample through receiver operating characteristic analysis." In: *Psychology of Addictive Behaviors* 28.1 (Mar. 2014) (cit. on p. 6).

[31] J. R. Quinlan. "Induction of decision trees". In: *Machine Learning* 1 (1 1986), pp. 81–106. DOI: https://doi.org/10.1007/BF00116251 (cit. on p. 53).

[32] Kristian Sikiric. "Gambling safety net: Predicting the risks of problem gambling using Bayesian networks". MA thesis. Linköping University, 2020. DOI: oai:DiVA.org:liu-165867 (cit. on p. 4).

[33] Ioannis I. Spyroglou et al. “Evaluation of Bayesian classifiers in asthma exacerbation prediction after medication discontinuation”. In: *BMC Research Notes* (2018). DOI: 10.1186/s13104-018-3621-1 (cit. on p. 7).

[34] Mohamed Wajdi Triki and Younes Boujelbene. “Bank credit risk : evidence from Tunisia using Bayesian networks”. eng. In: (2017) (cit. on p. 7).