Insights on Learning Tractable Probabilistic Graphical Models

Insights on Learning Tractable Probabilistic Graphical Models

Alvaro H.C. Correia

Insights on Learning Tractable Probabilistic Graphical Models
by Alvaro H.C. Correia.
Eindhoven: Technische Universiteit Eindhoven, 2023. Thesis.

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-5775-2

EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

SIKS Dissertation Series No. 2023-15
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords: Probabilistic Graphical Models, Bayesian Networks, Probabilistic Circuits, Machine Learning, Generative Models

Cover image: Iker Urteaga on Unsplash

Cover design: Alvaro H.C. Correia

Printed by: ADC Nederland

Insights on Learning Tractable Probabilistic Graphical Models

THESIS

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr. S.K. Lenaerts, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 22 juni 2023 om 16:00 uur

door

Alvaro Henrique Chaim Correia

geboren te Sao Caetano do Sul, Sao Paulo, Brazilië

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof.dr.ir. S.C. Borst
Promotor:	dr.habil. C. De Campos
Copromotor:	dr.ir. R. Peharz
Leden:	prof.dr.ir. B. de Vries
	dr.ir. J.H.P. Kwisthout (Radboud Universiteit)
	prof.dr. M. Zaffalon (Dalle Molle Institute for AI)
	prof.dr. M. Pechenizkiy

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Summary

This thesis focuses on the development of learning algorithms for probabilistic graphical models, notably Bayesian Networks and Probabilistic Circuits (PCs). The overarching theme of this research is exact inference with an emphasis on tractability, and is centred around the three main pieces of work described in the following.

For Bayesian Networks, we proposed new pruning techniques that reduce the computational cost of exact structure learning via the Bayesian Dirichlet Equivalent Uniform (BDeu) score. Exact structure learning in Bayesian Networks with BDeu is an NP-Hard problem that can be tackled by listing and scoring all the candidate parent sets for each variable. Since the BDeu is decomposable, the set of optimal structures can be computed exactly by searching through all possible combinations of candidate parent sets. The computational cost of this process can be significantly reduced if we can determine a priori which parent sets are not part of the optimal solution. That is where pruning methods come in. We propose a new upper bound on the BDeu score of supersets of a parent set, allowing us to discard any parent set whose upper bound is lower than the score of one of its subsets. Our upper bound is provenly tighter than previous upper bounds and, as we demonstrate experimentally, often reduces the search space by more than 50% in comparison to previous methods.

For Probabilistic Circuits, we proposed two different learning algorithms: one centred around structure learning and another around parameter learning. The first algorithm hinges on a connection between Probabilistic Circuits and Decision Trees (DTs). We show that DTs can be efficiently parsed to a PC while preserving the same underlying conditional distribution over the target vari-

able. That connection is fruitful for both PCs and DTs, since PCs can borrow discriminative structure learning methods from DTs, and DTs can be efficiently extended to model a full joint distribution over the input space. The result of this observation is a new class of models named Generative Forests: a natural extension of Random Forests that retains key characteristics Random Forests are known for (ease of training, little pre-processing needed, high accuracy), while bringing in a few capabilities only available to generative models, namely marginalisation of unobserved variables and outlier detection. In particular, our experiments show Generative Forests outperform popular imputation methods, like K-nearest neighbour imputation, on classification with missing data.

The second algorithm exploits continuous latent variables as a means to learn the parameters of Probabilistic Circuits. We draw inspiration from other generative models, like variational autoencoders (or VAEs), where a continuous latent space seems to facilitate learning. At a first glance, continuous latent variables are at odds with exact inference since we cannot marginalise them out exactly. However, for small latent dimensions numerical integration methods can get arbitrarily close to the true likelihood. Moreover, numerical integration methods actually approximate a continuous mixture via a discrete mixture, and the latter is naturally a tractable model (a PC in itself). In light of these observations, we propose continuous mixtures of tractable probabilistic models, which are essentially a mixture of PCs whose parameters are not independent, but connected via a continuous latent space. This simple approach shows surprisingly good results, outperforming all previous PC learning methods in a range of density estimation tasks.

List of Publications

The work described in Chapters 2-7 of this thesis has appeared in publications 1-7. The complete list of publications resulted from this PhD research is listed below.

Included Publications

(Correia and de Campos 2019) A. H. C. Correia, and C. de Campos, *Towards scalable and robust sum-product networks.*, International Conference on Scalable Uncertainty Management. Springer, Cham, 2019.

(Correia et al. 2019), A. H. C. Correia; C. de Campos, and L. C. van der Gaag, *An Experimental Study of Prior Dependence in Bayesian Network Structure Learning*, International Symposium on Imprecise Probabilities: Theories and Applications, 2019.

(Correia et al. 2020a), A. H. C. Correia; J. Cussens, and C. de Campos, *On pruning for score-based Bayesian network structure learning*, International Conference on Artificial Intelligence and Statistics (AISTATS). PMLR, 2020.

(Correia et al. 2020b), A. H. C. Correia; R. Peharz, and C. de Campos, *Joints in Random Forests*, Advances in neural information processing systems 33, 2020.

(Correia et al. 2023a), A.H.C. Correia; G. Gala; E. Quaeghebeur; C. de Campos, and R. Peharz, *Continuous Mixtures of Tractable Probabilistic Models*, to appear in Proceedings of the AAAI Conference on Artificial Intelligence. 37, (2023).

Non Included Publications

(Correia et al. 2018) A. H. C. Correia; J. L. M. Silva; T. C. Martins, and F. G. Cozman, *A fully attention-based information retriever*, 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018.

(Gusmão et al. 2018) A. C. Gusmão; A. H. C. Correia; G. de Bona, and F. G. Cozman, *Interpreting embedding models of knowledge bases: a pedagogical approach*, ICML Workshop on Human Interpretability in Machine Learning (WHI), 2018.

(Correia and Lecue 2019) A. H. C. Correia, and F. Lecue, *Human-in-the-loop Feature Selection*, Proceedings of the AAAI Conference on Artificial Intelligence. 33, 01 (Jul. 2019), 2438-2445.

(Correia et al. 2023b), A. H. C. Correia; D. W. Worrall, and R. Bondesan, *Neural Simulated Annealing*, to appear in International Conference on Artificial Intelligence and Statistics (AISTATS). PMLR, 2023.

Contents

List of Terms

AIC : Akaike Information Criterion.

BDeu : Bayesian Dirichlet equivalent uniform.

BIC : Bayesian Information Criterion.

BN : Bayesian Network.

BNSL : Bayesian Network Structure Learning.

CART : Classification and Regression Trees (Breiman et al. 1984).

CDF : Cumulative Distribution Function.

CLT : Chow-Liu Tree (Chow and Liu 1968).

CM : Continuous mixtures.

CNet : Cutset Network (Rahman et al. 2014).

DAG : Directed Acyclic Graph.

DT : Decision Tree (Breiman et al. 1984).

EiNet : Einsum Network (Peharz et al. 2020a).

ELBO : Evidence Lower Bound.

EM : Expectation Maximisation.

ESS : Equivalent Sample Size.

GAN : Generative Adversarial Network (Goodfellow et al. 2014).

GeDT : Generative Decision Tree.

GeF : Generative Forest.

H : Entropy.

KDE : Kernel Density Estimation.

KNN : K-Nearest Neighbours.

LBDeu : Local BDeu.

LLBDeu : Local-local BDeu.

LL : Log-likelihood.

LO : Latent Optimisation (Bojanowski et al. 2018).

MC : Monte Carlo.

MDL : Minimum Description Length.

MI : Mutual Information.

MLE : Maximum Likelihood Estimation.

PC : Probabilistic Circuit.

PDF : Probability Density Function.

QMC : Quasi-Monte Carlo.

RF : Random Forest (Breiman 2001).

RI : Robust Interval.

RQMC : Randomised Quasi-Monte Carlo.

SGD : Stochastic Gradient Descent.

SVHN : Street View House Numbers (Netzer et al. 2011).

UCI : University of California, Irvine.

VAE : Variational autoencoder (Kingma and Welling 2014).

List of Figures

List of Tables

Chapter 1

Introduction

The theme of this thesis is generative models, a class of machine learning models that, in the most simplistic and general definition, are capable of generating new samples. In recent years generative models powered by large and deep neural networks have achieved enormous success in text (Brown et al. 2020) and image generation (Ramesh et al. 2021), attracting vast research interest and surpassing most people's expectations on what current artificial intelligence technology can bring about. A variety of different models have contributed to this progress, with notable examples being variational autoencoders (VAEs) (Kingma and Welling 2014), generative adversarial networks (GANs) (Goodfellow et al. 2014), normalising flows (Papamakarios et al. 2021), and more recently diffusion models (Sohl-Dickstein et al. 2015; Ho et al. 2020), to name but a few. Yet, the common thread among all these methods is ever larger models which, despite producing impressive results and a sense of wonder at what deep neural networks can learn via simple gradient descent, are still poorly understood and offer little in terms of robustness and reliability.

This thesis focus on different classes of generative models that put probabilistic reasoning in the forefront, namely Bayesian Networks and Probabilistic Circuits. Albeit currently lagging behind deep generative models in most tasks, these models are principled and useful tools for probabilistic machine learning.

- Bayesian Networks (BNs) allow us to succinctly represent statistical and even causal dependencies among random variables. Thus, BNs are highly interpretable models that clearly lay out how our assumptions regarding one variable affect our predictions about another. We can design BNs to

satisfy constraints and relations known to hold in the real world or, perhaps more interestingly, infer such relations directly from data via structure learning algorithms, such as the ones studied in this thesis.

- While BNs are primarily representation tools, Probabilistic Circuits (PCs) are computation tools. PCs support exact and tractable marginal and conditional queries over arbitrary subsets of the domain variables. These queries form a consistent basis for probabilistic reasoning (Jaynes 2003), and hence enable us to solve a number of useful tasks reliably and consistently, like classification with missing data or data imputation.

In that context, the overarching motivation in this work is exactness and tractability. Any probabilistic machine learning model or algorithm give us a proper mathematical expression for any query or quantity of interest. For instance, we can always mathematically define a maximum a posteriori objective. However, this is only really useful if we can actually compute it and be reassured of a certain degree of precision. Otherwise, we will always be bound to question the trustworthiness of each prediction of our model, a prevailing problem with deep generative models. Therefore, the goal of our research is to study and improve tractable and exact learning algorithms and models. Concretely, we investigate how to improve the efficiency of exact structure learning algorithms for Bayesian Networks, and how to enhance the performance of Probabilistic Circuits both as classifiers, via connections to Decision Trees, and general purpose generative models, via the introduction of continuous latent variables. We formalise these ideas in the following research questions.

1.1 Research Questions

(Q1) **Can we still improve the computational efficiency of exact structure learning of BNs via pruning techniques?**

Exact structure learning of Bayesian Networks is typically marked by two steps, i) computing a score for each variable and each of its possible sets of parents, and ii) running a search algorithm to find the structure (combination of parent sets) of maximum total score. There are a number of pruning techniques in the literature, e.g. (de Campos and Ji 2011), that eliminate suboptimal parent sets from the search space, leading to significant gains in efficiency. We investigate whether we can derive even better pruning techniques by studying the properties of a popular scoring function, the Bayesian Dirichlet equivalent uniform (BDeu) score.

(Q2) **How to improve the performance of PCs in classification tasks?**

One typically learns the structure and parameters of Probabilistic Circuits to fit a joint distribution over the domain variables. That approach, however, more often than not, fails to match the performance of simpler discriminative models like decision trees. Can we devise new PC architectures or even borrow structure learning techniques from discriminative models to turn Probabilistic Circuits into more effective classifiers?

(Q3) **Can we leverage continuous latent variables in PCs?**

Continuous latent variables are an important ingredient in many powerful deep generative models, like VAEs, GANs and, to some extent, normalising flows. Nonetheless, Probabilistic Circuits are constrained to discrete latent variables since these can be integrated out efficiently, whereas continuous ones cannot. Yet, can we leverage continuous latent variables to facilitate parameter learning in Probabilistic Circuits?

(Q4) **How to quantify the robustness or reliability of PCs?**

One way to quantify the robustness of a prediction is to measure how much one can perturb the parameters of the model without changing its original prediction. This is the idea behind Credal PCs (Mauá et al. 2017; Maua et al. 2018), but unfortunately computing such robustness measures can be costly in general. Can we design PC architectures to facilitate these computations? How does that relate to accuracy, likelihood function and overall reliability of the model?

1.2 Thesis Contributions and Outline

In each chapter of this thesis we cover one contribution that we made in our effort to answer these research questions. Chapters 2 and 5 present the necessary background on Bayesian Networks and Probabilistic Circuits, respectively. These are, of course, brief expositions covering only a fraction of the literature on these topics, but they help contextualise our contributions and should be insightful for readers not entirely familiar with these two probabilistic models. The remaining chapters are outlined below.

1.2.1 Chapter 3

In this chapter we study pruning techniques for exact structure learning of Bayesian Networks. In a nutshell, exact structure learning requires computing scores for each possible parent set of each variable, and then running a search algorithm to find the structure that maximises the total sum of scores (a structure is fully defined by a collection of parent sets and vice-versa). Pruning works by discarding parent sets that have a subset of higher score. It is then useful to have upper bounds for these scores so as to prune entire regions of the search space without having to actually compute the score of every possible parent set. We focus on the Bayesian Dirichlet equivalent uniform (BDeu) score and show that, by exploiting properties of the gamma and likelihood functions, we were able to derive new tighter bounds for the BDeu score.

1.2.2 Chapter 4

We continue studying the properties of the BDeu score in Chapter 4, but this time we focus on how sensitive structure learning with the BDeu score is to the prior specified via the Equivalent Sample Size (ESS). Ideally, we would like the BDeu score to be prior-independent for large enough sample sizes, that is, the ESS should not influence the final learnt structure in the large-data regime. However, we have not observed that in our experiments and, even for relatively small numbers of variables, the amount of data required to make the BDeu score prior-independent was prohibitively large for most real-world scenarios.

1.2.3 Chapter 6

In Chapter 6 we move on to Probabilistic Circuits and investigate their connection to Decision Trees (DTs). It turns out that we can easily extend a DT to a PC by fitting joint distributions to each of its leaves. The resulting PC, which we call Generative Decision Trees or GeDT, is an interesting hybrid of discriminative and generative models, since its structure is learnt to minimise classification error, while its parameters are learnt to maximise the log-likelihood of the training data. Moreover, GeDTs preserve the same conditional distribution over the class variable as the original DTs but can also handle missing data in a principled and efficient manner via PC marginal queries. As we show in the experiments, GeDTs are effective classifiers and outperform popular techniques for treating missing data in DTs, like K-Nearest Neighbours (KNN) imputation.

1.2.4 Chapter 7

We also investigate the robustness of Probabilistic Circuits via their credal counterparts (Mauá et al. 2017), which are based on the theory of imprecise probabilities (Walley 1991). Credal PCs allow us to compute how much we can perturb their parameters without changing their predictions, which gives us a quantitative measure of robustness that we refer to as ϵ-robustness. In this chapter, we propose a novel PC architecture, named class-selective, that is more suitable for classification tasks and at the same time facilitates the computation of ϵ-robustness with respect to the target variable. We also ran experiments in a number of classification tasks to evaluate how ϵ-robustness relates to accuracy and log-likelihood in both class-selective PCs and GeDTs.

1.2.5 Chapter 8

In this chapter we look into ways to introduce continuous latent variables into Probabilistic Circuits. We propose a simple continuous mixture of PCs, where the parameters of each PC is dependent of the continuous latent variable via a learnable function parametrised by a neural network. These models are inherently intractable, but for small latent dimensions, can be accurately approximated via numerical integration methods, which allow us to train (via regular backprop) and run inference on these continuous mixture of PCs. Moreover, the numerical approximation actually yields a discrete mixture of PCs, which is a PC in itself. Thus, continuous latent variables can be seen as artifices to help us learn PCs, since we can first fit a continuous mixture model and then compile it into a PC via numerical integration. In our experiments, such continuous mixtures of PCs outperformed most existing PC models in the literature in a number of datasets, even though we only considered mixtures of PCs with extremely simple architectures, namely fully factorised distributions or Chow-Liu Trees (Chow and Liu 1968).

1.2.6 Chapter 9

Finally, we conclude this thesis in Chapter 9, where we summarise our main contributions and analyse their main limitations as well as possibly interesting directions for future research.

1.3 Research Reproducibility and Reuse

Each of the chapters is tied to a different contribution and therefore is also supported by a different source code. The relevant code repository is indicated in each of the chapters, but for the sake of completeness, we list all of them here.

Chapter 3 `https://github.com/alcorreia/bdeu-structure-learning`

Chapter 4 `https://github.com/alcorreia/bdeu-structure-learning`

Chapter 6 `https://github.com/alcorreia/gefs`

Chapter 7 `https://github.com/alcorreia/sum2019` and `https://github.com/alcorreia/gefs`

Chapter 8 `https://github.com/alcorreia/cm-tpm`

1.4 How to Read This Thesis

We tried to make the chapters of this thesis self-contained as much as possible. Each chapter briefly covers the necessary background and notation, and thus can be read on its own. Chapters 2 and 5 are general introductions to Bayesian Networks and Probabilistic Circuits, respectively, and readers familiar with those topics might skip these chapters entirely. Therefore, although we organised the chapters aiming for a gradual exposition of the topics we cover, this thesis can be read in any order.

Chapter 2

Bayesian Networks

Bayesian Networks are a class of generative models that graphically represent a joint distribution via a directed acyclic graph (the structure) and a set of parametric conditional distributions (the parameters). These are compact but powerful representations that clearly lay out statistical dependencies among variables. That makes Bayesian networks particularly useful when interfacing with users or experts, since their graphical nature exposes how variables affect one another and facilitates the injection of prior knowledge into the model. Moreover, we can learn these models directly from data, allowing us to uncover the statistical relationships among a set of variables via structure learning, which is the main focus of this thesis in what touches Bayesian Networks. In this chapter we present a general introduction to Bayesian Networks, with a focus on exact structure learning.

We begin our exposition with a general introduction to Bayesian Networks. This should provide the reader with the necessary background and notation to follow our first contributions. For a thorough introduction to Bayesian Networks we refer the reader to (Koller and Friedman 2009).

A Bayesian Network (Pearl 1988) is a widely used probabilistic graphical model encoding a probability distribution $P(\mathbf{X})$, where $\mathbf{X} = \{X_1, ..., X_m\}$ is a set of random variables of interest. It is composed of (i) a *structure* defined by a directed acyclic graph (DAG) $\mathcal{G}$ where each node is associated with a random variable X_i, and where arcs represent probabilistic dependencies entailing the *Markov* condition: every variable is conditionally independent of its non-descendant variables given its parents; and (ii) a collection of conditional probability distributions defined for each variable given its parents in the graph. More precisely, in the absence of missing data, the probability distribution defined by a Bayesian Network is given by

$$P(\mathbf{X}|\mathcal{G}, \boldsymbol{\theta}) = \prod_{i=1}^{m} P(X_i|\boldsymbol{\theta}_{X_i}, \mathrm{pa}(X_i)), \tag{2.1}$$

where the graph $\mathcal{G}$ factorises the *global distribution* $P(\mathbf{X}|\mathcal{G}, \boldsymbol{\theta})$ into independent *local distributions* $P(X_i|\,\mathrm{pa}(X_i), \boldsymbol{\theta}_{X_i})$, with $\boldsymbol{\theta}_{X_i}$ the parameters of the distribution over X_i, and $\mathrm{pa}(X_i)$ the parents of X_i in graph $\mathcal{G}$ (Pearl 1988). This factorisation is a central property of Bayesian Networks, enabling compact representations of complex probability distributions (Pearl 2014) as well as efficient learning algorithms, such as our work on structure learning (Correia et al. 2020a) discussed is Chapter 3.

A Bayesian Network $\mathcal{B}$ is completely defined by a tuple a $(\mathcal{G}, \boldsymbol{\theta})$, where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{X_i}\}_{i=1}^{m}$ collects the parameters of the local distributions, and $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ is the graph with one node (or vertex) for each variable and a set of *directed* edges $\mathbf{E}$ connecting them. In this work we use 'structure' and 'graph' interchangeably.

Their graphical nature is arguably the key feature of Bayesian Networks, facilitating the representation of complex probabilistic relationships existing in many real-world problems (Cussens et al. 2013). Moreover, it is the conditional independence relationships encoded in the graph that allow for an efficient and compact factorisation of the global distribution as in Equation 2.1. To be precise, two random variables X_i and X_j are said to be conditionally independent given a set of random variables $\mathbf{S}$ under a distribution P, if we have that $P(X_i, X_j \,|\, \mathbf{S}) = P(X_i \,|\, \mathbf{S})P(X_j \,|\, \mathbf{S})$[1]. In that context, it is useful to define a couple of relationships between graphs and distributions. We say a graph $\mathcal{G}$ *contains*

[1]Note that this definition ignores events of zero probability that may arise in practice.

a joint distribution P if P satisfies the conditional independencies induced by $\mathcal{G}$. Conversely, a distribution P is said to be *faithful* to a graph $\mathcal{G}$ if and only if all (and only those) independence facts true in P are entailed by the graphical structure $\mathcal{G}$.

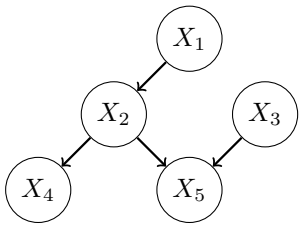

Figure 2.1: Example of a simple Bayesian Network defined over five random variables $\mathbf{X} = \{X_1, X_2, X_3, X_4, X_5\}$. The depicted graph induces the factorisation $P(\mathbf{X}) = P(X_1)P(X_2|X_1)P(X_3)P(X_4|X_2)P(X_5|X_2, X_3)$.

Furthermore, the graphical representation in Bayesian Networks greatly facilitates the interpretation of the relationship among variables and eases the integration of domain knowledge. In fact, in many cases the structure or parameters of a Bayesian Network are partially elicited from human experts (Buntine 1991; Heckerman et al. 1995; de Campos and Castellano 2007; Cano et al. 2011). In this thesis, however, we concentrate on data-driven methods where the goal is to learn the structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$ directly from some data $\mathcal{D}$. In that context, the learning task is commonly split between *structure learning*, which is the learning of independence relations among variables, and *parameter learning*, which is the learning of the individual distributions $P(X_i|\operatorname{pa}(X_i))$.

$$\underbrace{P(\mathcal{G}, \boldsymbol{\theta}|\mathcal{D})}_{\text{Learning}} = \underbrace{P(\mathcal{G}|\mathcal{D})}_{\text{Structure learning}} \cdot \underbrace{P(\boldsymbol{\theta}|\mathcal{D})}_{\text{Parameter learning}}$$

In this thesis, we focus primarily on the structure learning problem since that is where our contributions lie. It is also worth mentioning that in the case of discrete data (which we assume in the context of Bayesian Networks), finding the optimal model reduces to finding the optimal structure: if data $\mathcal{D}$ is complete and discrete, optimal parameters, in the maximum-likelihood sense, are readily given by frequency counts from the data (Silander et al. 2008). Therefore, in the rest of this chapter we delve into different approaches to structure learning

that have been proposed in the literature. However, before continuing we must first briefly introduce some notation and technicalities regarding the data.

As mentioned previously, a Bayesian Network is defined over a fixed set of variables denoted $\mathbf{X} = \{X_1, X_2, \ldots, X_m\}$, which we assume to be drawn from a fixed joint distribution $\mathbb{P}^*(\mathbf{X})$. While we do not have access to the true distribution $\mathbb{P}^*$, we assume that we have a dataset $\mathcal{D}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of n independent and identically distributed (i.i.d.) samples from $\mathbb{P}^*$, where $\mathbf{x}$ is a joint instantiation of variables $\mathbf{X}$ and $\mathbf{x}[i]$ is the state in $\mathbf{x}$ belonging to variable X_i. In what touches Bayesian Networks, the work in this thesis assumes discrete random variables, and thus, we denote the domain of a variable X_i as $\mathcal{X}_i = \{1, \ldots, K_i\}$, where K_i is the number of states of X_i. Correspondingly, the set of all joint instantiations of a set of random variables $\mathbf{S} \subseteq \mathbf{X}$, i.e. $\boldsymbol{\mathcal{X}}_{\mathbf{S}}$, is the Cartesian product of the state space of the individual variables, $\boldsymbol{\mathcal{X}}_{\mathbf{S}} = \bigtimes_{X_i \in S} \mathcal{X}_i$. We use $q(\boldsymbol{\mathcal{X}}_{\mathbf{S}}) = |\boldsymbol{\mathcal{X}}_{\mathbf{S}}|$ to denote the size of state space $\boldsymbol{\mathcal{X}}_{\mathbf{S}}$ and, with a slight abuse of notation, $q(i) = |\mathcal{X}_i| = K_i$.

In Bayesian network structure learning (BNSL) with discrete data, we also need to keep track of how many observations we get of each possible configuration of a variable and its parent set in a given graph, which unfortunately requires some extra, rather cumbersome notation. For a fixed graph $\mathcal{G}$, we use n_{ijk} to denote the number of observations in $\mathcal{D}$ with $X_i = k$ and $\mathrm{pa}(X_i) = j$, and $n_{ij} = \sum_k n_{ijk}$. Note that in this notation, k takes values in $\{1, \ldots, K_i\}$ and j takes values in $\{1, \ldots, q(\mathrm{pa}(X_i))\}$. We use a similar notation for the parameters $\boldsymbol{\theta}$ of a Bayesian Network. We denote the probability $P(X_i = k \mid \mathrm{pa}(X_i) = j)$ as θ_{ijk} and use $\boldsymbol{\theta}_{ij}$ to denote the union of the k parameters θ_{ijk}, i.e. $\boldsymbol{\theta}_{ij} = \{\theta_{ijk} \mid k \in \{1, \ldots, K_i\}\}$.

2.1 Structure Learning

Bayesian Network structure learning is the problem of searching for a graph $\mathcal{G}$ that fits some observed data $\mathcal{D}$ well with respect to desired criteria. There are two main classes of approaches to BNSL in the literature:

- **Score-based** approaches rely on a scoring function that quantifies how well a given graph fits the data. A search algorithm is used to navigate the space of graphs looking for the graph of highest score.

- **Constraint-based** infer the presence (or absence) of edges in a graph via statistical conditional independence tests.

There are a number of effective constraint-based approaches, the PC algorithm (Spirtes et al. 2000; Colombo et al. 2014) being a notable example, but we do not cover these methods in this text. Instead, we will dive into different score-based methods proposed in the literature, since our contributions target this class of approaches. In particular, we are interested in the properties of scoring functions, notably the Bayesian Dirichlet equivalent uniform (BDeu) score, and less so on the different search algorithms and heuristics found in the literature. Therefore, in the rest of this chapter we will focus on and introduce the most popular scoring functions for structure learning.

Before moving on and introducing score-based BNSL methods, we first formalise a couple of assumptions about the data $\mathcal{D}$ that will underpin our discussion on structure learning.

Assumption 1 (Samples from 'true Bayesian Network')
The data $\mathcal{D}$ consists of independent and identically distributed (i.i.d.) samples from a (unknown) Bayesian Network $\mathcal{B}^* = (\mathcal{G}^*, \boldsymbol{\theta}^*)$, where each random variable X_i, with parents $\mathrm{pa}(X_i)^*$ in the true graph $\mathcal{G}^*$, follows a multinomial distribution

$$X_i \mid \mathrm{pa}(X_i)^* \sim \textit{Multinomial}(\boldsymbol{\theta}^*_{ij} \mid \mathrm{pa}(X_i)^*).$$

We refer to $\mathcal{B}^*$ and $\mathcal{G}^*$ as the true Bayesian Network and the true graph, respectively. The goal of structure learning is often framed as recovering the true graph $\mathcal{G}^*$, which is indeed the best we can hope for if we want to understand the domain structure. Note that Assumption 1 is not restrictive since Bayesian Networks can represent any joint distribution in finite discrete domains, which encompass all applications of Bayesian Networks considered in this thesis.

Assumption 2 (Complete data)
All observations in $\mathcal{D}$ are complete. That means that for every $\mathbf{x} \in \mathcal{D}$ and every $i \in \{1, \ldots, m\}$, we observe $X_i = \mathbf{x}[i]$.

Incomplete data is still problematic in structure learning and very few approaches actually tackle this scenario; see for instance (Friedman 1998) and (Adel and De Campos 2017). Thus, most approaches, including all discussed in this thesis, assume complete data.

2.1.1 Score-based Approaches

In this class of structure learning methods, we use a scoring function to measure the goodness of fit of a given structure to the data. Learning is then a search

problem aiming at finding the graph (or sets of graphs) of maximum score. Unfortunately, learning DAGs from data is not an easy computational problem. It is known to be NP-hard (Chickering 1996; Chickering et al. 2004), and the large search space[2] makes brute-force methods unfeasible in most interesting applications. Thus, we often resort to search heuristics, like A^* search (Yuan et al. 2011), or other optimisation methods (Zheng et al. 2018). Yet, exact algorithms (de Campos et al. 2009; Cussens 2011) have gained traction recently thanks to increases in computational power and improvements in learning algorithms, like the bounds we propose in the next chapter (Correia et al. 2020a).

Regardless of how we navigate the search space of possible DAGs, we would like our scoring functions to satisfy a few properties to facilitate the search and ensure the resulting DAGs are reasonable representations of the statistical information provided in the data. In the following we introduce five such desiderata common to many popular scoring functions.

Desideratum 1 (Decomposability)
A scoring function is decomposable if the score of a Bayesian Network $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$ can be computed as the sum of the scores of individual variables in accordance to the factorisation induced by the graph $\mathcal{G}$ as in Equation 2.1

$$score(\mathcal{G} \mid \mathcal{D}) = \sum_{i=1}^{m} score(X_i \mid \mathrm{pa}(X_i), \mathcal{D}).$$

As we will see in the next chapter, decomposability greatly facilitates structure learning, since it allows us to study the search space in a per-variable basis, resulting in more efficient and effective pruning techniques. In a nutshell, if a graph is known to be suboptimal for variable X_i than it is also suboptimal over the entire scope $\mathbf{X}$ and can be discarded early on.

One central question in BNSL is whether we can indeed hope to recover the true $\mathcal{G}^*$. Before addressing this question, we need to introduce the concept of equivalency among Bayesian Network structures.

Definition 1 (Equivalent structures (Chickering 1995))
Two directed acyclic graphs $\mathcal{G}$ and $\mathcal{G}'$ are equivalent if for every Bayesian Network $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$, there is $\mathcal{B}' = (\mathcal{G}', \boldsymbol{\theta}')$ such that $\mathcal{B}$ and $\mathcal{B}'$ define the same probability distribution.

This simply tells us that two graphs are equivalent if they can encode the same joint probability distributions. A set of equivalent DAGs is commonly re-

[2]The number of valid DAGs is super-exponential in the number of variables $|\mathbf{X}|$ (Robinson 1977).

ferred to as an *equivalence class*. Definition 1 is often called *distribution equivalence* (Chickering 2002), but under Assumption 1 of multinomial random variables, we could equally use a different notion of equivalence and define it as the representation of the same set of conditional independencies, usually referred to as *Markov equivalence* (Madigan et al. 1996; Richardson 1997) or *I-equivalence* (Koller and Friedman 2009). However, the definition above more clearly shows a direct and important consequence of equivalence among graphs: without further assumptions or information on the relationship among variables, we cannot hope to learn a unique graph from data. Even if we had access to an infinite number of samples from the true distribution, two equivalent structures offer the exact same statistical support to the observed data, and thus cannot be distinguished by data-dependent scores alone. We would like our scoring functions to reflect this limitation, which leads us to our next desideratum.

Desideratum 2 (Score equivalence (Chickering 1995))
The scores of two equivalent Bayesian Network structures must be equal.

We would also like to have guarantees that our scoring function converges to the true graph. That is, in the limit of infinite data, we would like the true graph to be the maximiser of our scoring function. That reassures us that, if we have enough data and score equivalence, the graph of maximum score will be the true graph or belong to the same equivalence class as the true graph. That is captured in the desideratum of consistency.

Desideratum 3 (Consistency (Chickering 2002))
A scoring function is consistent if, in the limit of infinite samples from a given distribution P, i.e. $n = |\mathcal{D}| \to \infty$, the following two conditions hold.

- *If graph $\mathcal{G}$ contains P and $\mathcal{G}'$ does not*[3]*, then score$(\mathcal{G} \mid \mathcal{D}) >$ score$(\mathcal{G}' \mid \mathcal{D})$.*
- *If both $\mathcal{G}$ and $\mathcal{G}'$ contain P, but $\mathcal{G}$ has fewer parameters than $\mathcal{G}'$, then score$(\mathcal{G} \mid \mathcal{D}) >$ score$(\mathcal{G}' \mid \mathcal{D})$.*

With that we are ready to introduce the first class of score-based methods that concern Bayesian scoring functions.

2.1.2 Bayesian scoring functions

Our contributions in the field of Bayesian Networks rely on Bayesian scoring functions, so we go in depth about these scores here. Bayesian scoring functions

[3]Remember that we say a graph $\mathcal{G}$ *contains* a joint distribution P if P satisfies the conditional independencies induced by $\mathcal{G}$.

are designed to approximate the posterior, i.e. the probability of the graph given the data

$$\underbrace{P(\mathcal{G} \mid \mathcal{D})}_{\text{posterior}} = \frac{\overbrace{P(\mathcal{D}|\mathcal{G})}^{\text{likelihood}}\overbrace{P(\mathcal{G})}^{\text{prior}}}{P(\mathcal{D})}.$$

In the context of Bayesian inference, BNSL is a maximum a posteriori (MAP) query, where we search for the graph that maximises the posterior $P(\mathcal{G} \mid \mathcal{D})$. Since the data probability (or normalisation constant) $P(\mathcal{D}) = \sum_{\mathcal{G}} P(\mathcal{D}|\mathcal{G})P(\mathcal{G})$ is intractable for most interesting problems and does not influence model selection (does not depend on $\mathcal{G}$), Bayesian scores are designed to compute only $P(\mathcal{D}|\mathcal{G})P(\mathcal{G})$. Without loss of generality, we can express Bayesian scores in log space[4] as

$$\text{score}_{\text{Bayes}}(\mathcal{G} \mid \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}). \tag{2.2}$$

The Bayesian scores of the form (2.2) that we consider in this thesis are decomposable and proven to be consistent (Chickering 2002). However, a few of them are not score equivalent as we will see shortly.

As usual we can incorporate user knowledge via the prior $P(\mathcal{G})$. For instance, we could exclude a number of structures based on some known causal relationship among the variables. However, in the rest of this thesis, we will assume no such information is available and will use a uniform prior. In that case, all structures are just as likely a priori, and the scores can be fully described via the log-likelihood function. However, the log-likelihood of a given graph $\mathcal{G}$ also depends on parameters $\boldsymbol{\theta}$, which are not known a priori. Thus, in Bayesian inference, we compute the log-likelihood by defining a prior over the parameters $P(\boldsymbol{\theta} \mid \mathcal{G})$ and integrating over the parameter space

$$\text{LL}^{\text{mar}}(\mathcal{D} \mid \mathcal{G}) = \log P(\mathcal{D} \mid \mathcal{G}) = \int_{\boldsymbol{\theta}} \log P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) \underbrace{P(\boldsymbol{\theta} \mid \mathcal{G})}_{\text{parameter prior}} d\boldsymbol{\theta}. \tag{2.3}$$

We refer to the function in Equation 2.3 as the *marginal log-likelihood*, or simply $\text{LL}^{\text{mar}}(\mathcal{D} \mid \mathcal{G})$, since it requires marginalising out the parameters $\boldsymbol{\theta}$. In general, the marginal log-likelihood is intractable due to the integral in (2.3), which forms a hard computational problem. Thus, we need a few extra assumptions on top of Assumptions 1 and 2 to be able to compute it in closed form.

[4]The choice of the base of the logarithm is immaterial (as long as used consistently) and we often omit it in this thesis to ease the notation. Most common bases are 2 and Euler's number e, and the use of one or the other should be clear by the units: nats for base e and bits for base 2.

Assumption 3 (Parameter independence)
The prior over parameters $P(\boldsymbol{\theta} \mid \mathcal{G})$ satisfies parameter independence*:*

$$P(\boldsymbol{\theta} \mid \mathcal{G}) = \prod_i \prod_j P(\boldsymbol{\theta}_{ij} \mid \mathcal{G}).$$

Assumption 4 (Parameter modularity)
The prior over parameters $P(\boldsymbol{\theta} \mid \mathcal{G})$ satisfies parameter modularity*: if X_i has the same set of parents in two different graphs $\mathcal{G}$ and $\mathcal{G}'$ then*

$$P(\boldsymbol{\theta}_{ij} \mid \mathcal{G}) = P(\boldsymbol{\theta}_{ij} \mid \mathcal{G}').$$

Assumption 4 is convenient because it ensures the prior over $\boldsymbol{\theta}_{ij}$ depends only on the local structure of the graph immediately surrounding X_i.

Assumption 5 (Dirichlet distribution)
The prior over parameters $P(\boldsymbol{\theta} \mid \mathcal{G})$ has a Dirichlet distribution

$$P(\boldsymbol{\theta}_{ij} \mid \mathcal{G}) \propto \prod_k \theta_{ijk}^{\alpha_{ijk}-1},$$

with $\alpha_{ijk} > 0$ a user specified parameter for each θ_{ijk}.

Due to the conjugacy between the Dirichlet and categorical distributions, α_{ijk} can be interpreted as pseudo-counts, i.e. they represent the number of counts of each category that we expect to observe or, rather, that we have observed in our prior experience, before running the current experiment.

With these assumptions, we have the following closed-form expression for the marginal log-likelihood that gives rise to the **Bayesian Dirichlet (BD) score** (Cooper and Herskovits 1992; Heckerman et al. 1995)

$$\begin{aligned} \mathrm{BD}(\mathcal{G}) = \sum_{i=1}^{m} \sum_{j=1}^{q(\mathrm{pa}(X_i))} \Bigg[\log \left(\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \right) + \\ \sum_{k=1}^{q(X_i)} \log \left(\frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \right) \Bigg]. \end{aligned} \tag{2.4}$$

Unfortunately, the BD score is not often useful in practice since it requires specifying all of the numerous terms α_{ijk}. One solution is to assign $\alpha_{ijk} = 1$ for all

i, j and k, which is equivalent to (uninformative) zero pseudo-counts. That is the so-called **K2 score** (Cooper and Herskovits 1992)

$$\text{K2}(\mathcal{G}) = \sum_{i=1}^{m} \sum_{j=1}^{q(\text{pa}(X_i))} \left[\log \left(\frac{(q(X_i) - 1)!}{(q(X_i) - 1 + n_{ij})!} \right) + \sum_{k=1}^{q(X_i)} \log \left(n_{ijk}! \right) \right].$$

The K2 score is already much easier to use, since we no longer have to define individual values for α_{ijk}. However, the K2 score is not score equivalent. In fact, not every set of pseudo-counts α_{ijk} satisfy score equivalence, thus arbitrary configurations of the BD score also fall short of Desideratum 2. To recover this property, we need to establish constrains on the parameters α_{ijk}.

Theorem 1 (Bayesian Dirichlet equivalence (Heckerman et al. 1995))
Two Markov equivalent graphs with equal prior $P(\mathcal{G})$ have the same Bayesian score if and only if

$$\alpha_{ijk} = \alpha \cdot P(X_i = k, pa(X_i) = j \mid \mathcal{G}),$$

*where α is a scalar commonly referred to as the **equivalent sample size** (or **ESS**) and $P(X_i = k, pa(X_i) = j \mid \mathcal{G})$ is a prior probability.*

Theorem 1 gives us a principled way to set the parameters α_{ijk} and ensure score equivalence. The resulting score, commonly called **Bayesian Dirichlet equivalence (BDe) score**, is still hard to use in practice since it requires defining $P(X_i = k, pa(X_i) = j \mid \mathcal{G})$ for every configuration of i, j, k. A solution is to define a uniform prior with

$$P(X_i = k, pa(X_i) = j \mid \mathcal{G}) = \frac{1}{q(X_i)q(\text{pa}(X_i))}.$$

This choice of prior for the BDe score gives the so-called **Bayesian Dirichlet equivalent uniform (BDeu) score**, which was theoretically justified in (Heckerman et al. 1995), but had already been proposed in (Buntine 1991) and (Cooper and Herskovits 1992). The BDeu is computed as in Equation 2.4 and only requires setting the equivalent sample size (ESS) α, with the individual pseudo-counts given by

$$\alpha_{ijk} = \frac{\alpha}{q(X_i)q(\text{pa}(X_i))},$$

which by Theorem 1 guarantees score equivalence.

Despite the apparent ease of use of the BDeu score—it requires setting only a single hyperparameter—there is no consensus on how one should choose an appropriate value for the ESS α. Yet, the ESS has a significant impact on the resulting MAP structure as shown in (Silander et al. 2007) and our own work (Correia et al. 2019) discussed in Chapter 4.

2.1.3 Penalised (Maximum) Log-Likelihood Scores

In the last section, we have relied on the marginal likelihood function to find MAP structures. However, we could also have considered maximum-likelihood estimates (MLE) of the parameters $\boldsymbol{\theta}$ and approximate the log-likelihood as

$$\mathrm{LL}^{\mathrm{mle}}(\mathcal{G} \mid \mathcal{D}) = \log P(\mathcal{D}|\mathcal{G}, \boldsymbol{\theta}^{\mathrm{mle}}). \tag{2.5}$$

Assuming complete and discrete data, the maximum-likelihood estimate $\boldsymbol{\theta}^{\mathrm{mle}}$ corresponds to frequency counts from the data (Silander et al. 2008)

$$\theta^{\mathrm{mle}}_{ijk} = \hat{P}(X_i = k \mid \mathrm{pa}(X_i) = j) = \frac{n_{ijk}}{n_{ij}},$$

where we use $\hat{P}$ to emphasise this is a empirical distribution. For the sake of conciseness, we will refer to the function in Equation 2.5 simply as *maximum log-likelihood* and denote it $\mathrm{LL}^{\mathrm{mle}}(\mathcal{G} \mid \mathcal{D})$.

Given complete and i.i.d. data $\mathcal{D}$, we have the following expression for the maximum log-likelihood (Bouckaert 1995)

$$\begin{aligned}\mathrm{LL}^{\mathrm{mle}}(\mathcal{G} \mid \mathcal{D}) &= \sum_{i=1}^{m} \sum_{j=1}^{q(\mathrm{pa}(X_i))} \sum_{k=1}^{q(X_i)} n_{ijk} \log \hat{P}(X_i = k \mid \mathrm{pa}(X_i) = j, \boldsymbol{\theta}^{\mathrm{mle}}) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{q(\mathrm{pa}(X_i))} \sum_{k=1}^{q(X_i)} n_{ijk} \log \left(\frac{n_{ijk}}{n_{ij}} \right), \end{aligned} \tag{2.6}$$

where $\hat{P}$ is the distribution defined by the Bayesian Network $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta}^{\mathrm{mle}})$, and we combine all n_{ijk} occurrences of each parameter $\theta^{\mathrm{mle}}_{ijk} = n_{ijk}/n_{ij}$.

It is clear from Equation 2.6 that the maximum log-likelihood is decomposable, but unfortunately it is not consistent since it fails to satisfy the second condition of Desideratum 3 requiring that graphs with fewer parameters be assigned a higher score. That is because adding an extra edge to a DAG never decreases its maximum log-likelihood, and thus a scoring function based solely

on the maximum log-likelihood will always favour complete graphs. To recover consistent scoring functions we have to add a penalisation term that reduces the score of a graph in proportion to its number of parameters. There are a number of ways one can define such penalisation, but all maximum log-likelihood scores we discuss in this chapter assume the following form

$$\text{score}_{\text{LL}}(\mathcal{G} \mid \mathcal{D}) = \text{LL}^{\text{mle}}(\mathcal{G} \mid \mathcal{D}) - \sum_{i=1}^{m} \text{Pen}(X_i, \mathcal{G}, \mathcal{D}), \tag{2.7}$$

with Pen a penalisation term that is also decomposable. We can break down the penalisation term as a function of the number of parameters used to represent the distribution of each variable, i.e. $|\boldsymbol{\theta}_{X_i}|$,

$$\text{Pen}(X_i, \mathcal{G}, \mathcal{D}) = f(n)|\boldsymbol{\theta}_{X_i}| = f(n)(q(i) - 1)q(\text{pa}(X_i)).$$

The last equality stems from the assumption of discrete data: for each of the $q(\text{pa}(X_i))$ possible instantiations of $\text{pa}(X_i)$, we require $q(i) - 1$ parameters to represent the distribution over X_i[5]. The function $f : \mathbb{R} \mapsto \mathbb{R}$ controls the strength of the penalisation term and is what varies among the different penalised log-likelihood scores.

Notable examples are the **Akaike Information Criterion** (**AIC**) with $f(n) = 1$ and the **Bayesian Information Criterion** (**BIC**) with $f(n) = \log(n)/2$. The BIC score is particularly interesting because it satisfies our three desiderata: it is decomposable as all scores of the form (2.7), it is score equivalent (Scutari 2018) and consistent (Haughton 1988). In particular, it can be shown that, for Bayesian Networks over multinomial data, Bayesian scores as in Equation 2.2 converge to the BIC score as the number of datapoints n approaches infinity (Haughton 1988; Geiger et al. 2001). This theoretically justifies the BIC as a scoring function as well as motivates its use as tool to study the asymptotic behaviour of Bayesian scores (Slobodianik et al. 2009).

The BIC score can also be motivated by an information theory perspective (Akaike 1998; Suzuki 1999), as it is equivalent to the **Minimum Description Length** (**MDL**) score. The MDL score aims at measuring how well a model represents the data $\mathcal{D}$ into a code of minimum length composed of two parts:

- The minimum number of bits required to encode the data. This term captures how well the model explains the data, the so-called goodness of fit, and is exactly the maximum log-likelihood.

[5] The $q(i)^{\text{th}}$ parameter is fully specified given the other $q(i)-1$ ones since they define a categorical distribution and thus, must sum to one.

- The minimum number of bits required to represent the model itself. That is the penalisation term favouring shorter codes, i.e. models with small number of parameters. Since the expected number of bits required to represent a probability value is $\log(n)/2$, we have $f(n) = \log(n)/2$.

We can extend the information theory perspective to maximum log-likelihoods scores at large, which gives further insigths into this class of scores and facilitates the introduction of Chow-Liu Trees (Chow and Liu 1968) later on. For that we rely on the concepts of empirical mutual information and entropy functions (Koller and Friedman 2009)

$$\mathrm{LL}^{\mathrm{mle}}(\mathcal{G} \mid \mathcal{D}) = n \sum_{i=1}^{m} \mathrm{MI}_{\hat{P}}(X_i, \mathrm{pa}(X_i)) - n \sum_{i=1}^{m} \mathrm{H}_{\hat{P}}(X_i), \tag{2.8}$$

where we use $\hat{P}$ to emphasise we compute these terms with respect to the empirical distribution, and we have the usual definitions of the mutual information MI and entropy H with respect to a discrete distribution P

$$\begin{aligned} \mathrm{MI}_P(\mathbf{X}, \mathbf{Y}) &= \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \sum_{\mathbf{y} \in \boldsymbol{\mathcal{Y}}} P(\mathbf{x}, \mathbf{y}) \log \frac{P(\mathbf{x}, \mathbf{y})}{P(\mathbf{x}) P(\mathbf{y})} \\ \mathrm{H}_P(\mathbf{X}) &= - \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} P(\mathbf{x}) \log P(\mathbf{x}), \end{aligned} \tag{2.9}$$

with $\mathbf{X}$ and $\mathbf{Y}$ arbitrary discrete random vectors. The mutual information is a metric that captures the mutual dependence between random variables: the higher the mutual information the more dependent two variables are. Equation 2.8 then gives us the valuable insight that the marginal likelihood measures the strength of the dependencies between variables and its parents in a given graph, and thus is a useful signal for structure learning.

Moreover, Equation 2.8 decomposes the marginal likelihood in a term that depends on the structure, the mutual information, and another that is independent of the structure, the entropy. This makes it evident that the difference between the marginal likelihood of two graphs only depends on the mutual information, which is one way to demonstrate the marginal log-likelihood is not consistent. Since the mutual information is a metric, and thus non-negative, adding an edge to a graph will also add non-negative terms to the right-hand side of Equation 2.8 and will never decrease the maximum log-likelihood. To see this note that adding an edge amounts to increasing the parent set of a given random variable, and thus augmenting the state space we sum over when computing the mutual information as in Equation 2.9.

Chow-Liu Trees (CLTs)

One common design choice in structure learning is to limit the search space to trees, that is, structures where each node (variable) has no more than one parent. Trees are simple structures with arguably limited representation power, since we are constrained on the types of conditional independence we can capture; no collider nodes, for instance.

Yet, tree structures are popular because they facilitate exact inference[6] and we have principled and efficient algorithms to learn tree structures from data. A notable example that will be relevant in this thesis is the so-called Chow-Liu Tree (Chow and Liu 1968), a tree-shaped Bayesian Network that directly minimises the Kullback-Leibler (KL) divergence (Kullback and Leibler 1951) between the true data distribution $\mathbb{P}^*(\mathbf{X})$ and the distribution represented by the Bayesian Network $P(\mathbf{X})$

$$D_{\mathrm{KL}}(\mathbb{P}^* || P) = \sum_{\mathbf{x}} \mathbb{P}^*(\mathbf{x}) \log\left(\frac{\mathbb{P}^*(\mathbf{x})}{P(\mathbf{x})}\right).$$

Chow and Liu (Chow and Liu 1968) proved that the tree-shaped graph that minimises such KL divergence can be found by searching for a maximum weight spanning tree in a fully-connected graph where nodes are in one-to-one correspondence with variables $\mathbf{X}$, and the weight of an edge is the mutual information between the pair of variables it connects. This is equivalent to a score-based approach maximising the $\mathrm{LL}^{\mathrm{mle}}(\mathcal{G} \mid \mathcal{D})$, since the entropy term in Equation 2.8 is independent of the graph. However, it has the advantage of admitting exact polynomial time solution, $\mathcal{O}(m^2)$ to be precise, and not requiring any penalisation, as we are already restricted to the space of tree-shaped graphs where there is little risk of overfitting.

In summary, an algorithm to learn a Chow-Liu tree from data $\mathcal{D}$ can be decomposed into three main steps.

1. Compute the mutual information for every pair of variables in $\mathbf{X}$ and construct a complete *undirected* graph where an edge between variables X and Y has weight given by $\mathrm{MI}_{\hat{P}}(X, Y)$, where $\hat{P}$ is the empirical distribution induced by data $\mathcal{D}$.

2. Find a maximum weight spanning tree over the graph defined in the previous step with e.g. Kruskal's or Prim's algorithm (Cormen et al. 2022).

[6] In Bayesian Networks with tree-shaped graphs, marginal and conditional probabilities can be computed in time linear in the number of nodes.

3. Select a root node among $\mathbf{X}$ and assign directions to the edges in the maximum weight spanning tree, with arcs pointing outward the root node.

The Chow-Liu Tree algorithm is a useful and time-tested structure learning method that is relevant not only in the field of Bayesian Networks but also in the Probabilistic Circuits literature as we shall see in Chapter 5 (Rahman et al. 2014; Liu and Van den Broeck 2021). We also use Chow-Liu Trees in our own work on Probabilistic Circuits that we discuss in Chapter 8.

2.2 Conclusion

We have presented a brief introduction to Bayesian Networks along with the scoring functions and learning algorithms that are the most relevant to our work. The concepts discussed in this chapter should be sufficient to follow the main contributions in this thesis, but this is not a comprehensive exposition on the topic. We refer the reader to (Koller and Friedman 2009) for a more thorough overview of Bayesian Networks and Probabilistic Graphical Models in general.

Chapter 3

On pruning for score-based Bayesian network structure learning

Many algorithms for score-based Bayesian network structure learning (BNSL), in particular exact ones, take as input a collection of potentially optimal parent sets for each variable in the data. Constructing such collections naively is computationally intensive since the number of parent sets grows exponentially with the number of variables. Thus, pruning techniques are not only desirable but essential. While good pruning rules exist for the Bayesian Information Criterion (BIC), current results for the Bayesian Dirichlet equivalent uniform (BDeu) score reduce the search space very modestly, hampering the use of the (often preferred) BDeu. In this chapter, we derive new non-trivial theoretical upper bounds for the BDeu score that considerably improve on the state of the art. Since the new bounds are mathematically proven to be tighter than previous ones and at little extra computational cost, they are a promising addition to BNSL methods.

This chapter is almost integrally based on Alvaro Correia, James Cussens and Cassio de Campos: On pruning for score-based Bayesian network structure learning, *International Conference on Artificial Intelligence and Statistics (AISTATS), 2709-2718*, 2020.

3.1 Introduction

As discussed in the previous chapter, a Bayesian network (Pearl 1988) is a popular probabilistic graphical model given by a directed acyclic graph (DAG) $\mathcal{G}$ and a set of parameters $\boldsymbol{\theta}$ defining conditional probability distributions. In this chapter, we focus on learning $\mathcal{G}$ from data and introduce new bounds that facilitate *exact* Bayesian network structure learning (BNSL). When using *score-based* approaches to BNSL, such bounds are extremely useful to prune areas of the search space that are known not to contain the optimal graph, considerably speeding up the optimisation.

In particular, we focus on the *Bayesian Dirichlet equivalent uniform* (BDeu) score (Buntine 1991; Cooper and Herskovits 1992; Heckerman et al. 1995), which consists in the log probability of the graph given (multinomial) data and a uniform prior on structures. See Chapter 2 for an overview of Bayesian scoring functions and a more thorough introduction to the BDeu score.

Of particular interest in this work is the fact that the BDeu score is *decomposable*. That is, we can write the BDeu of a graph $\mathcal{G}$, whose nodes are in correspondence with a set of m random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$, as a sum of *local scores*, one for each variable X_i:

$$\mathrm{BDeu}(\mathcal{G}) = \sum_{i=1}^{m} \mathrm{LBDeu}(X_i, \mathrm{pa}(X_i)), \tag{3.1}$$

where LBDeu is the local BDeu score function and $\mathrm{pa}(X_i)$ are the parents of variable X_i in the graph $\mathcal{G}$.

A common approach to *exact* BNSL divides the problem into two steps:

1. *Candidate Parent Set Identification*: For each variable, find a suitable collection of candidate parent sets and their local scores.

2. *Structure Optimisation*: Given the collection of candidate parent sets, choose a parent set for each variable so as to maximise the overall score while avoiding directed cycles.

This chapter concerns pruning ideas to help solve candidate parent set identification. Simply put, we aim at reducing the number of BDeu scores we have to compute by discarding parent sets that will not lead to an optimal solution at the second step. In turn this also reduces the search space and hopefully running times of structure optimisation algorithms.

BNSL is known to be NP-hard (Chickering et al. 2004) and the subproblem of parent set identification is unlikely to admit a polynomial-time (in n) algorithm; it is proven to be LOGSNP-Hard for BIC (Koivisto 2006). As a compromise, one typically chooses a maximum in-degree d (number of parents per node) and computes the score only for parent sets with in-degree at most d. Naturally, that does reduce the search space but comes at the cost of discarding numerous potentially optimal graphs. Conversely, increasing the maximum in-degree can considerably improve the chances of finding better structures but requires higher computing time: there are $\Theta(n^d)$ candidate parent sets (per variable) if an exhaustive search is performed with in-degree d, and 2^{n-1} without any in-degree constraint. The large search space is an important limiting factor in exact BNSL, as $d > 2$ is already prohibitively expensive for many interesting applications (Bartlett and Cussens 2017).

Our goal is then to prune this search space more aggressively to help scale exact BNSL with BDeu. We provide new theoretical upper bounds for the local scores that allow us to identify and discard non-optimal parent sets without ever having to compute their scores. These new upper bounds are efficient and can be readily integrated into any search approach (Chen et al. 2016; Cussens 2011; de Campos and Ji 2011; de Campos et al. 2009; Jaakkola et al. 2010; Koivisto and Sood 2004; Yuan and Malone 2012, 2013).

While our study has been motivated by the scientific interest in solving the BNSL problem in an exact manner, we shall note that local scores for a variable given its parents also have a probabilistic interpretation (the decomposition of the score comes from independence assumptions). Therefore, new approaches to prune such search space of parent sets can be useful for other purposes too.

This chapter is organised as follows. Section 3.2 provides the notation and required definitions, as well as a brief description of the current best bound for BDeu in the literature, which we call ub_f. Section 3.4 presents a new improved bound ub_g whose derivation follows the same mathematical approach as the existing state-of-the-art bound but further exploits properties of the score function to get better results. This new bound is provably tighter than previous ones but still does not capture all cases and other bounds can be devised. Section 3.5 looks at the problem from a new angle and introduces a bound ub_h based on a (tweaked) maximum likelihood estimation. Bounds ub_g and ub_h leverage different aspects of the problem and we show how they can be effectively combined for an even more aggressive pruning in Section 3.6. Finally, Section 3.9 concludes the chapter and gives directions for future research.

3.2 Definitions and Notation

Before presenting the main results, we have to introduce some notation that differs from the rest of the thesis in a few points. First of all, in an attempt to lighten the notation, we often denote variables (nodes of a Bayesian Network) by their indices (e.g. representing X_i simply as i). Also, since the collection of scores are computed independently for each variable (BDeu is decomposable) and the main results do not dependent on specific variables, we simplify the notation and drop the i from Equation 3.1, using simply $\mathrm{LBDeu}(\mathbf{S})$ to refer to the score of node i with parent set $\mathbf{S}_i \subseteq \mathbf{X}^{\setminus i}$ in the DAG $\mathcal{G}$, with $\mathbf{X}^{\setminus i} = \mathbf{X} \setminus \{i\}$.

We still need some further notation:

- The state space of variable X_i is denoted by $\mathcal{X}_i$. Similarly, $\boldsymbol{\mathcal{X}}_{\mathbf{S}}$ is the set of all joint instantiations of the random variables in $\mathbf{S} \subseteq \mathbf{X}$, that is, $\boldsymbol{\mathcal{X}}_{\mathbf{S}}$ is the Cartesian product of the state space of involved variables, with $\boldsymbol{\mathcal{X}}_{\mathbf{S}} = \bigtimes_{X_i \in S} \mathcal{X}_i$. We denote the size of state space $\boldsymbol{\mathcal{X}}_{\mathbf{S}}$ as $q(\mathbf{S}) = |\boldsymbol{\mathcal{X}}_{\mathbf{S}}|$, and we abuse notation to say $q(i) = |\mathcal{X}_i|$.

- We reserve i for (indices of) variables and j for instances of a state space, e.g., $j_{\mathbf{S}} \in \boldsymbol{\mathcal{X}}_{\mathbf{S}}$. The subscript is omitted if clear from the context.

- The data $\mathcal{D}$ is a *multiset* (repetitions are allowed) of elements from $\boldsymbol{\mathcal{X}}$, with $\mathcal{D}^{\mathbf{S}}$ the projection of $\mathcal{D}$ onto variables $\mathbf{S} \subseteq \mathbf{X}$ (note that $\mathcal{D}{=}\mathcal{D}^{\mathbf{X}}$). The same notation applies to projections of instantiations, e.g. $j^{\mathbf{S}}$. Moreover, we use $\mathcal{D}^{\mathbf{S}}(j_{\mathbf{S}'}) \subseteq \mathcal{D}^{\mathbf{S}}$ to denote the elements of $\mathcal{D}^{\mathbf{S}}$ compatible with a given $j_{\mathbf{S}'} \in \boldsymbol{\mathcal{X}}_{\mathbf{S}'}$, that is, $\mathcal{D}^{\mathbf{S}}(j_{\mathbf{S}'}) = \{j_{\mathbf{S}} : j_{\mathbf{S}} \in \mathcal{D}^{\mathbf{S}}, j_{\mathbf{S}}^{\mathbf{S}\cap\mathbf{S}'} = j_{\mathbf{S}'}^{\mathbf{S}\cap\mathbf{S}'}\}$. Finally, we use $\mathcal{D}_u$ instead of $\mathcal{D}$ to denote the set of unique elements from a given multiset $\mathcal{D}$.

- For an instantiation $j \in \boldsymbol{\mathcal{X}}_{\mathbf{S}}$, we define $n_j = |\mathcal{D}^{\mathbf{S}}(j)|$, that is, the number of occurrences of instantiation j in $\mathcal{D}^{\mathbf{S}}$. With a slight abuse of notation, for a given variable i we have $n_j = \sum_{k \in \{1,\ldots,q(i)\}} n_{jk}$.

- The vector $\vec{\alpha}_j = (\alpha_{jk})_{k \in \mathcal{X}_i}$ is the prior for parent set $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ under configuration $j \in \boldsymbol{\mathcal{X}}_{\mathbf{S}}$. In the BDeu score, $\vec{\alpha}_j$ satisfies $\alpha_{jk} = \alpha_{\text{ess}}/q(\mathbf{S}\cup\{i\})$, where α_{ess} is the equivalent sample size, a pre-defined parameter setting the strength of the prior. We also denote $\sum_{k \in \mathcal{X}_i} \alpha_{jk} = \alpha_{\text{ess}}/q(\mathbf{S})$ by α_j.

- Let $\Gamma_\alpha(x) = \frac{\Gamma(x+\alpha)}{\Gamma(\alpha)}$ for x non-negative integer and $\alpha > 0$ (Γ denotes the Gamma function).

3.2.1 Local BDeu Score

We are now ready to the define the local score $\mathrm{LBDeu}(\mathbf{S})$. For a variable i with parent set $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$, the local score can be written as

$$\mathrm{LBDeu}(\mathbf{S}) = \sum_{j \in \boldsymbol{\mathcal{X}}_{\mathbf{S}}} \mathrm{LLBDeu}(\mathbf{S}, j), \text{ and}$$
$$\mathrm{LLBDeu}(\mathbf{S}, j) = -\log \Gamma_{\alpha_j}(n_j) + \sum_{k \in \mathcal{X}_i} \log \Gamma_{\alpha_{jk}}(n_{jk}) \ .$$

$\mathrm{LBDeu}(\mathbf{S})$ is a sum of $q(\mathbf{S})$ values each of which is specific to a particular instantiation of variables in S. We call such values *local local BDeu scores (llB)*. In particular, $\mathrm{LLBDeu}(\mathbf{S}, j) = 0$ if $n_j = 0$, so we concentrate on instantiations j that do appear in the data:

$$\mathrm{LBDeu}(\mathbf{S}) = \sum_{j \in \mathcal{D}_u^{\mathbf{S}}} \mathrm{LLBDeu}(\mathbf{S}, j) \, .$$

This formula does not come by chance. In Section 3.5 we discuss its relation with the posterior probability of having $\mathbf{S}$ as parent of i.

3.3 Pruning in Candidate Parent Set Identification

The pruning of parent sets rests on the (simple) observation that a parent set cannot be optimal if one of its subsets has a higher score (Teyssier and Koller 2005). Thus, when learning Bayesian networks from data using BDeu, it is useful to have an upper bound

$$\mathrm{ub}(\mathbf{S}) \geq \max_{\mathbf{T}:\mathbf{T} \supset \mathbf{S}} \mathrm{LBDeu}(\mathbf{T}) \tag{3.2}$$

so as to potentially prune a whole area of the search space at once. Ideally, one would like an upper bound that is both tight (with respect to the inequality in Expression 3.2) and cheap to compute, so that one can score parent sets incrementally, and at the same time check whether it is worth 'expanding' them: if $\mathrm{ub}(\mathbf{S})$ is not greater than $\max_{\mathbf{R}:\mathbf{R} \subseteq \mathbf{S}} \mathrm{LBDeu}(\mathbf{R})$, then it is unnecessary to expand S. Figure 3.1 illustrates how a hypothetical bound would prune the search space.

With that in mind, we can define candidate parent set identification more formally.

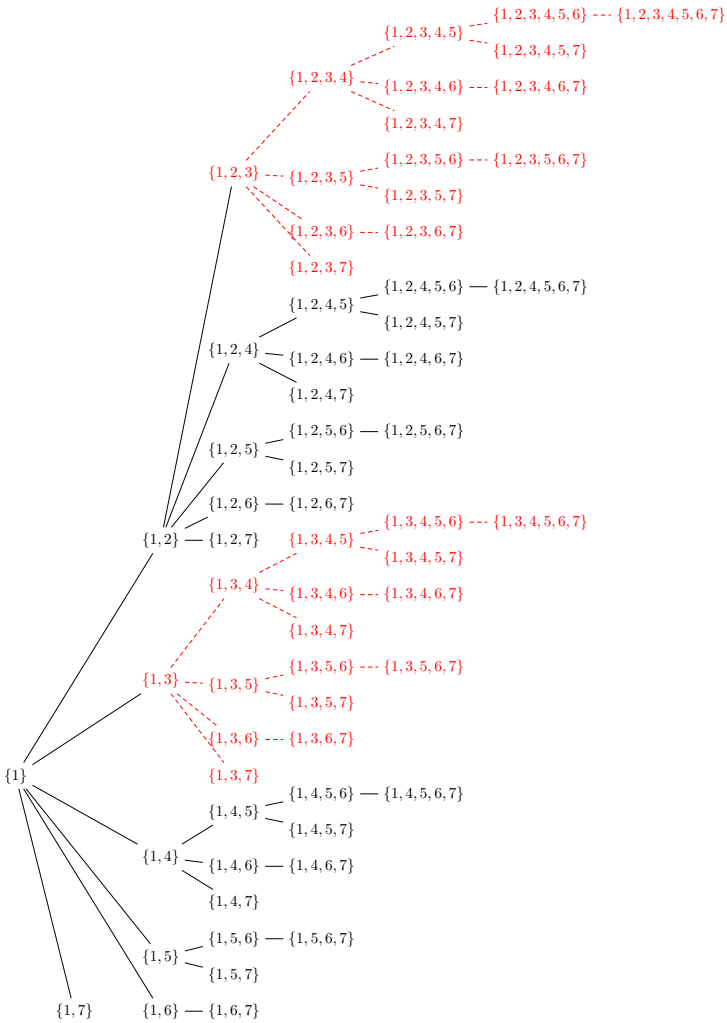

Figure 3.1: Illustration of potential parent sets in a dataset with 8 variables (the 8th variable is the child in this example and does not show). This is still a small part of the search space with only sets including variable 1. In red dashed lines, the sets pruned if $\mathrm{LBDeu}(\{1\}) \geq \mathrm{ub}(\{1,3\})$.

Definition 2 (Candidate Parent Set Identification)
For each variable $i \in \mathbf{X}$, find a collection of parent sets

$$\mathcal{L}_i = \{\mathbf{S} \subseteq \mathbf{X}^{\setminus i} : \mathbf{S}' \subset \mathbf{S} \Rightarrow \mathrm{LBDeu}(\mathbf{S}') < \mathrm{LBDeu}(\mathbf{S})\}\,.$$

Unfortunately, we cannot predict the elements of $\mathcal{L}_i$ and have to compute the scores for a list L_i potentially much larger than $\mathcal{L}_i$. The practical benefit of our bounds is to reduce $|L_i|$, thus lowering the computational cost of BNSL, while ensuring we do not miss any potentially optimal parent set, that is, we ensure $L_i \supseteq \mathcal{L}_i$. Before presenting the current best bound in the literature (Cussens 2012; de Campos and Ji 2010, 2011), we give a lemma on the variation of counts with expansions of parent sets.

Lemma 1
For $\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{X}^{\setminus i}$, $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$ and $j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}$ with $j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}$, we have

$$|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}| \geq |\mathcal{D}_u^{\mathbf{S}\cup\{i\}}|, \;\; \textit{and} \;\; |\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})| \leq |\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})|.$$

Proof. Given that $\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{X}^{\setminus i}$, every instantiation in $\mathcal{D}_u^{\mathbf{S}\cup\{i\}}$ is compatible with one or more elements of $\mathcal{D}_u^{\mathbf{T}\cup\{i\}}$, and thus $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}| \geq |\mathcal{D}_u^{\mathbf{S}\cup\{i\}}|$. The relationship is reversed when we consider unique occurrences compatible with a given instantiation. By construction $j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}$, so if there is an instantiation $j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}$, there must be at least one corresponding $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$, and it follows that $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})| \leq |\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})|$. Note that both $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})|$ and $|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})|$ are bounded by $q(i)$: one instantiation for each value child i can assume. □

As an example, consider the small dataset of Table 3.1. The number of non-zero counts never decreases as we add a new variable to the parent set of variable $i = 3$. With $\mathbf{S} = \{1\}$ and $\mathbf{T} = \{1, 2\}$, we have $|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}| = 3$ and $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}| = 4$. Conversely, the number of (unique) occurrences compatible with a given instantiation of the parent set never increases with its expansion: for example with $j_{\mathbf{S}} = (1)$ and $j_{\mathbf{T}} = (1, 1)$, we have $|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})| = 2$ and $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})| = 2$.

We now introduce function f that, for a variable i, is defined on the sets of potential parents $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$, and observed instantiations $j \in \mathcal{D}_u^{\mathbf{S}}$:

$$f(\mathbf{S}, j) = -|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j)| \log q(i)\,,$$
$$f(\mathbf{S}) = \sum_{j \in \mathcal{D}_u^{\mathbf{S}}} f(\mathbf{S}, j).$$

Table 3.1: Example of data $\mathcal{D}$, its reductions by parent sets $\mathbf{S} = \{1\}$ and $\mathbf{T} = \{1, 2\}$, and the number of unique occurrences compatible with $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$ and $j_{\mathbf{T}}, j'_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}$, with $j_{\mathbf{T}}^{\mathbf{S}} = j'^{\mathbf{S}}_{\mathbf{T}} = j_{\mathbf{S}}$. The child variable is $i = 3$, and we have $j_{\mathbf{S}} = (1)$, $j_{\mathbf{T}} = (1, 1)$, $j'_{\mathbf{T}} = (1, 0)$.

$\mathcal{D}$

1	2	3
0	0	0
1	0	0
1	1	0
1	1	1

$\mathcal{D}_u^{\mathbf{S}\cup\{i\}}$

1	3
0	0
1	0
1	1

$\mathcal{D}_u^{\mathbf{T}\cup\{i\}}$

1	2	3
0	0	0
1	0	0
1	1	0
1	1	1

$\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})$

1	3
1	0
1	1

$\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})$

1	2	3
1	1	0
1	1	1

$\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j'_{\mathbf{T}})$

1	2	3
1	0	0

Theorem 2 (ub_f)
For a variable i, a potential parent set $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ and its instantiations $j \in \mathcal{D}_u^{\mathbf{S}}$, we have that

$$\mathrm{LLBDeu}(\mathbf{S}, j) \leq f(\mathbf{S}, j).$$

Moreover, if $\mathrm{LBDeu}(\mathbf{S}') \geq \sum_{j \in \mathcal{D}_u^{\mathbf{S}}} f(\mathbf{S}, j) = f(\mathbf{S})$ for some $\mathbf{S}' \subset \mathbf{S}$, then all $\mathbf{T} \supseteq \mathbf{S}$ are not in $\mathcal{L}_i$ (Cussens and Bartlett 2015; de Campos and Ji 2011).

From Theorem 2, we get an upper bound on the local BDeu score of all supersets of parent set $\mathbf{S}$

$$\mathrm{ub}_f(\mathbf{S}) = f(\mathbf{S}) \geq \max_{\mathbf{T}:\mathbf{T}\supset\mathbf{S}} \mathrm{LBDeu}(\mathbf{T}).$$

In words, we compute the number of non-zero counts per instantiation, $|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j)|$, and we 'gain' $\log q(i)$ for each of them. It is worth noticing that $f(\mathbf{S}) = -|\mathcal{D}_u^{\mathbf{S}\cup\{i\}}| \log q(i)$, which by Lemma 1 is monotonically non-increasing over expansions of the parent set $\mathbf{S}$. Hence $f(\mathbf{S})$ is not only an upper bound on $\mathrm{LBDeu}(\mathbf{S})$ but also on $\mathrm{LBDeu}(\mathbf{T})$ for every $\mathbf{T} \supseteq \mathbf{S}$. Bound ub_f is cheap to compute but is unfortunately too loose. We derive much tighter upper bounds on $\mathrm{LLBDeu}(\mathbf{S}, j)$ (where $n_j > 0$) by considering instantiation counts for the *full* parent set $\mathbf{X}^{\setminus i}$, the parent set that includes all possible parents for child i. We call these *full instantiation counts*. Evidently, the number of full parent instantiations $q(\mathbf{X}^{\setminus i})$ grows exponentially with $|V|$, but it is linear in $|\mathcal{D}|$ when we consider only the unique elements $\mathcal{D}_u^{\mathbf{X}^{\setminus i}}$.

3.4 Exploiting the Gamma Function

First, we extend the current state-of-the-art upper bound of Theorem 2 by exploiting some properties of the Gamma function. For that, we need some intermediate results, where we assume $\alpha > 0$.

Lemma 2
Let x be a positive integer. Then

$$\Gamma_\alpha(0) = 1 \quad \textit{and} \quad \log \Gamma_\alpha(x) = \sum_{\ell=0}^{x-1} \log(\ell + \alpha)\,.$$

Proof. By definition $\log(\Gamma_\alpha(x)) = \log\left(\frac{\Gamma(x+\alpha)}{\Gamma(\alpha)}\right)$, so the first statement follows directly: $\log(\Gamma_\alpha(0)) = \log\left(\frac{\Gamma(0+\alpha)}{\Gamma(\alpha)}\right) = \log(0) = 1$. Since it is a property of the gamma function that $\Gamma(x+1) = x\Gamma(x)$ for any positive real number x, we can compute $\log(\Gamma_\alpha(1)) = \log\left(\frac{\Gamma(1+\alpha)}{\Gamma(\alpha)}\right) = \log\left(\frac{\alpha\Gamma(\alpha)}{\Gamma(\alpha)}\right) = \log(\alpha)$, which is in accordance with the second statement. Finally, we can recursively compute

$$\begin{aligned}
\log(\Gamma_\alpha(x)) &= \log\left(\frac{\Gamma(x-1+\alpha+1)}{\Gamma(\alpha)}\right) \\
&= \log\left(\frac{(x-1+\alpha)\Gamma(x-1+\alpha)}{\Gamma(\alpha)}\right) \\
&= \log(x-1+\alpha) + \log(\Gamma_\alpha(x-1)),
\end{aligned}$$

and since x is a positive integer, we can repeat this step until reaching the base case $\Gamma_\alpha(0)$, with

$$\log \Gamma_\alpha(x) = \sum_{\ell=0}^{x-1} \log(\ell+\alpha) + \underbrace{\log(\Gamma_\alpha(0))}_{=\log(1)=0} = \sum_{\ell=0}^{x-1} \log(\ell+\alpha)$$

□

Lemma 3
For x positive integer and $v \geq 1$,

$$\log\left(\frac{\Gamma_\alpha(x)}{\Gamma_{\alpha/v}(x)}\right) \geq \log v\,.$$

Proof. By applying Lemma 2, we obtain

$$\sum_{\ell=0}^{x-1} \log \frac{\ell+\alpha}{\ell+\alpha/v} = \log v + \sum_{\ell=1}^{x-1} \log \frac{\ell+\alpha}{\ell+\alpha/v} \geq \log v\,,$$

as each term of the sum (if any) is greater than zero. □

Lemma 4
Let x, y be non-negative integers such that $x+y>0$. Then

$$\begin{cases} \Gamma_\alpha(x+y) = \Gamma_\alpha(x)\,\Gamma_\alpha(y) & \text{if } x\cdot y = 0\,, \\ \Gamma_\alpha(x+y) \geq \Gamma_\alpha(x)\,\Gamma_\alpha(y)\,(1+y/\alpha) & \text{otherwise.} \end{cases}$$

Proof. If x (resp. y) is zero, then $\Gamma_\alpha(x) = 1$ and the equality holds. Otherwise we apply Lemma 2 three times and manipulate the products:

$$\begin{aligned} \frac{\Gamma_\alpha(x+y)}{\Gamma_\alpha(x)\,\Gamma_\alpha(y)} &= \frac{\prod_{z=0}^{x+y-1}(z+\alpha)}{\prod_{z=0}^{x-1}(z+\alpha)\prod_{z=0}^{y-1}(z+\alpha)} \\ &= \prod_{z=y}^{x+y-1}(z+\alpha)\prod_{z=0}^{x-1}\frac{1}{(z+\alpha)} \\ &= \prod_{z=0}^{x-1}\frac{y+z+\alpha}{z+\alpha} \geq \frac{y+\alpha}{\alpha}\,, \end{aligned}$$

which holds since all terms in this final product are greater or equal to 1. □

Corollary 1
Let $x_1, \ldots, x_k$ be a list of non-negative integers in decreasing order with $x_1 > 0$, then

$$\Gamma_\alpha\left(\sum_{l=1}^{k} x_l\right) \geq \prod_{l=1}^{k}\Gamma_\alpha(x_l)\prod_{l=1}^{k'-1}(1+x_l/\alpha)\,,$$

where $k' \leq k$ is the last positive integer in the list (in this notation, the second product on the right-hand side disappears if $k'=1$).

Proof. Repeatedly apply Lemma 4 to $x_t + (\sum_{l=t}^{k} x_l)$ until all elements are processed. While both the current x_t and the rest of the list are positive (until $t = k'-1$), we obtain the extra term $(1+x_t/\alpha)$. After that, we only 'collect' the Gamma functions of the first product on the right-hand side, so the result follows. □

Lemma 5
For $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ and $j \in \mathcal{D}_u^{\mathbf{S}}$, assume that $\vec{n}_j = (n_{jk})_{k \in \mathcal{X}_i}$ are in decreasing order over $k = 1, \ldots, q(i)$ (this is without loss of generality, since we can name and process them in any order). Then for any $\alpha \geq \alpha_j = \alpha_{ess}/q(\mathbf{S})$, we have

$$\begin{aligned}\mathrm{LLBDeu}(\mathbf{S}, j) \leq & f(\mathbf{S}, j) + g(\mathbf{S}, j, \alpha), \\ g(\mathbf{S}, j, \alpha) = & -\sum_{l=1}^{k'-1} \log\left(1 + n_{j,l}/\alpha\right),\end{aligned}$$

where $k' \leq k$ is the largest index such that $n_{j,k'} > 0$.

Proof. First of all, we have

$$\begin{aligned}\mathrm{LLBDeu}(\mathbf{S}, j) = & -\log \Gamma_{\alpha_j}(n_j) + \sum_{k \in \mathcal{X}_i} \log \Gamma_{\alpha_{jk}}(n_{jk}) \\ = & -\log \Gamma_{\alpha_j}\left(\sum_{k \in \mathcal{X}_i} n_{jk}\right) + \sum_{k \in \mathcal{X}_i} \log \Gamma_{\alpha_{jk}}(n_{jk}) .\end{aligned}$$

Since counts n_{jk} are in decreasing order by k, we apply Corollary 1:

$$\begin{aligned}\mathrm{LLBDeu}(\mathbf{S}, j) &\leq -\log\left(\prod_{l=1}^{q(i)} \Gamma_{\alpha_j}(n_{j,l}) \prod_{l=1}^{k'-1} (1 + \frac{n_{j,l}}{\alpha_j})\right) + \sum_{k \in \mathcal{X}_i} \log \Gamma_{\alpha_{jk}}(n_{jk}) \\ &= \sum_{k \in \mathcal{X}_i} \log\left(\frac{\Gamma_{\alpha_{jk}}(n_{jk})}{\Gamma_{\alpha_j}(n_{jk})}\right) - \sum_{l=1}^{k'-1} \log\left(1 + \frac{n_{j,l}}{\alpha_j}\right) \\ &\leq -|\mathcal{D}_u^{\mathbf{S} \cup \{i\}}(j)| \log q(i) - \sum_{l=1}^{k'-1} \log\left(1 + \frac{n_{j,l}}{\alpha}\right)\end{aligned}$$

where $\alpha \geq \alpha_j$ and we have by Lemma 3 that $\Gamma_{\alpha_{jk}}(n_{jk}) / \Gamma_{\alpha_j}(n_{jk}) \leq -\log q(i)$ whenever $n_{jk} > 0$. □

The difference here is the summation from the gap of the super-multiplicativity of Γ (Lemma 4 and Corollary 1). That extra term gives us a tighter bound on $\mathrm{LLBDeu}(\mathbf{S}, j)$, but $g(\mathbf{S}) = f(\mathbf{S}) + \sum_{j \in \mathcal{D}_u^{\mathbf{S}}} g(\mathbf{S}, j, \alpha)$ is no longer monotonic over expansions of $\mathbf{S}$ (albeit monotone in α). Hence, $g(\mathbf{S})$ is not an upper bound on $\mathrm{LBDeu}(\mathbf{T})$ for every $\mathbf{T} \supseteq \mathbf{S}$, and we need further results on $g(\mathbf{S}, j, \alpha)$.

Lemma 6
For $\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{X}^{\setminus i}$*,* $j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}$*, and* $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$ *with* $j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}$*, we have*

$$f(\mathbf{T}, j_{\mathbf{T}}) \geq f(\mathbf{S}, j_{\mathbf{S}}),$$

$$g(\mathbf{T}, j_{\mathbf{T}}, \alpha) \geq g(\mathbf{S}, j_{\mathbf{S}}, \alpha).$$

Proof. Because $j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}$, by Lemma 1 we have $|\mathcal{D}_u^{\mathbf{T}\cup\{i\}}(j_{\mathbf{T}})| \leq |\mathcal{D}_u^{\mathbf{S}\cup\{i\}}(j_{\mathbf{S}})|$. Moreover, $n_{j_{\mathbf{T}},k} \leq n_{j_{\mathbf{S}},k}$ for every $k \in \mathcal{X}_i$ (the counts get partitioned as more parents are introduced to arrive at $\mathbf{T}$ from $\mathbf{S}$), so $(1 + n_{j_T,k}/\alpha) \leq (1 + n_{j_{\mathbf{S}},k}/\alpha)$ for every k, and the result follows. □

Using this property of g as described in Lemma 6, we can pick the best value of g over all full expansions j of a current instantiation $j_{\mathbf{S}}$ to create a valid bound:

Theorem 3 (ub$_g$)
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$*,* $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$*, Then*

$$\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) \leq f(\mathbf{S}, j_{\mathbf{S}}) + \underline{g}(\mathbf{S}, j_{\mathbf{S}})$$

$$\underline{g}(\mathbf{S}, j_{\mathbf{S}}) = \min_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}:\ j^{\mathbf{S}} = j_{\mathbf{S}}} g(\mathbf{X}^{\setminus i},\ j,\ \alpha_{ess}/q(\mathbf{S}))$$

Also, if $\mathrm{LBDeu}(\mathbf{S}') \geq (f(\mathbf{S}) + \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{g}(\mathbf{S}, j_{\mathbf{S}})) = \underline{g}(\mathbf{S})$ *for some* $\mathbf{S}' \subset \mathbf{S}$*, then all* $\mathbf{T} \supseteq \mathbf{S}$ *are not in* $\mathcal{L}_i$*.*

Proof. First we prove that $f(\mathbf{S}, j_{\mathbf{S}}) + \underline{g}(\mathbf{S}, j_{\mathbf{S}})$ is an upper bound for $\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}})$. From Lemma 6, if we take any instantiation of the fully expanded parent set, $j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} : j^{\mathbf{S}} = j_{\mathbf{S}}$, we have that $g(\mathbf{S}, j_{\mathbf{S}}, \alpha) \leq g(\mathbf{X}^{\setminus i}, j, \alpha)$ for any α. As Lemma 6 is valid for every full instantiation j, we take the minimum over them to get the tightest bound. From Lemma 5, $\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) \leq f(\mathbf{S}, j_{\mathbf{S}}) + \underline{g}(\mathbf{S}, j_{\mathbf{S}})$. Now, if we sum all the llBs, we obtain the second part of the theorem for $\mathbf{S}$.

Finally, we need to show that this second part of the theorem holds for any $\mathbf{T} \supset \mathbf{S}$, which follows from $f(\mathbf{T}) \leq f(\mathbf{S})$ (as the total number of non-zero counts only increases, by Lemma 1) and

$$\begin{aligned}\sum_{j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}} \underline{g}(\mathbf{T}, j_{\mathbf{T}}) &= \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \left(\sum_{j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}:\ j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}} \underline{g}(\mathbf{T}, j_{\mathbf{T}}) \right) \\ &\leq \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{g}(\mathbf{S}, j_{\mathbf{S}})\,.\end{aligned}$$

That holds as $\underline{g}(\mathbf{T}, j_{\mathbf{T}}) \leq 0$ and, with $j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}$, at least one term $\underline{g}(\mathbf{T}, j_{\mathbf{T}})$ is smaller than $\underline{g}(\mathbf{S}, j_{\mathbf{S}})$, as their minimisation spans the same full instantiations (and $g(\cdot, \cdot, \alpha)$ is non-decreasing on α). □

In brief, the relevance of Theorem 3 is that it gives us a tighter upper bound $\text{ub}_g(\mathbf{S}) \leq \text{ub}_f(\mathbf{S})$, such that

$$\begin{aligned}\text{ub}_g(\mathbf{S}) = \underline{g}(\mathbf{S}) &= (f(\mathbf{S}) + \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{g}(\mathbf{S}, j_{\mathbf{S}})) \\ &\geq \max_{\mathbf{T}:\mathbf{T}\supset\mathbf{S}} \text{LBDeu}(\mathbf{T}).\end{aligned}$$

Therefore, this bound is always equal or superior to the current state-of-the-art bound at the time of writing. Moreover, the overhead of computing the bounds is negligible if a smart implementation is used (one that reuses computations that are nevertheless required for calculating the scores). The process which constructs contingency tables of counts for local score computations (say, from an AD-tree) is the main bottleneck in scoring, but it can be cheaply extended to simultaneously produce tables of sets of 'full instantiations' for the computation of upper bounds where, for instance, addition of counts are replaced with unions of sets. While this technical detail is irrelevant for the mathematical proofs here, it is important to point out that the new bounds imply very little extra computational costs.

3.5 Exploiting the Likelihood Function

Bound ub_g of previous section was based on the best full instantiation $j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}$ that is compatible with an llB of the parent set $\mathbf{S}$. Knowing that function g is monotonic over parent set sizes, we could look at an instantiation of the fully extended parent set to derive a bound for the llB of $\mathbf{S}$ and all its supersets. Even though the results are valid for every full instantiation, we can only compute bound ub_g using one of them at a time. The new bound of this section comes from the realisation that it is possible to exploit all full instantiations to derive a valid bound on the llB of $\mathbf{S}$. For that purpose, we need some properties of inference with the Dirichlet-multinomial distribution and conjugacy.

The BDeu score is simply the log marginal probability of the observed data given suitably chosen Dirichlet priors over the parameters of a BN structure. Consequently, llBs are intimately connected to the Dirichlet-multinomial conjugacy. Given a Dirichlet prior $\vec{\alpha}_j = (\alpha_{j,1}, \ldots, \alpha_{j,q(i)})$, the probability of observing

data $\mathcal{D}_{\vec{n}_j}$ with counts $\vec{n}_j = (n_{j,1}, \ldots, n_{j,q(i)})$ is:

$$\log \Pr(\mathcal{D}_{\vec{n}_j}|\vec{\alpha}_j) = \log \int_p \Pr(\mathcal{D}_{\vec{n}_j}|p)\Pr(p|\vec{\alpha}_j)dp\,,$$

where the first distribution under the integral is multinomial and the second is Dirichlet. Note that

$$\log \int_p \Pr(\mathcal{D}_{\vec{n}_j}|p)\Pr(p|\vec{\alpha}_j)dp \leq \max_p \log \Pr(\mathcal{D}_{\vec{n}_j}|p), \tag{3.3}$$

since $\int_p \Pr(p|\vec{\alpha}_j)dp = 1$. Note also that llBs are not the probability of observing sufficient statistics counts, but of a particular dataset, that is, there is no multinomial coefficient which would consider all the permutations yielding the same sufficient statistics. Therefore, we may devise a new upper bound based on the maximum (log-)likelihood estimation.

Lemma 7
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ *and* $j \in \mathcal{D}_u^{\mathbf{S}}$. *Then*

$$\begin{aligned} &\mathrm{LLBDeu}(\mathbf{S}, j) \leq \mathit{ML}(\vec{n}_j) \\ &\textit{with } \mathit{ML}(\vec{n}_j) = \sum_{k \in \mathcal{X}_i} n_{jk} \log(n_{jk}/n_j)^1. \end{aligned}$$

Proof. The llB is simply the log probability of observing a data sequence with counts $\vec{n}_j$ under a Dirichlet-multinomial distribution with parameter vector $\vec{\alpha}_j$. The result follows from Expression equation 3.3 and holds for any prior $\vec{\alpha}_j$. □

Corollary 2
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ *and* $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$. *Then*

$$\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) \leq \sum_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}:\, j^{\mathbf{S}} = j_{\mathbf{S}}} \mathit{ML}(\vec{n}_j)\,.$$

Proof. This follows from the properties of the maximum likelihood estimation, because it is monotonically non-decreasing with the expansion of parent sets (in terms of maximum likelihood, we fit the distribution just as well or better when having more parents). □

[1] In this notation, we use $0 \log 0 = 0$.

We can improve further on this bound of Corollary 2 by considering llBs as a function h of α for fixed $\vec{n}_j$, since we can study and exploit the shape of their curves. We define

$$h_{\vec{n}_j}(\alpha) = -\log \Gamma_\alpha(n_j) + \sum_{k \in \mathcal{X}_i} \log \Gamma_{\alpha/q(i)}(n_{jk}) \ .$$

Lemma 8
If $\nexists k : n_{jk} = n_j$, then $h_{\vec{n}_j}$ is a concave function for positive $\alpha \leq 1$.

Proof. (This result can also be obtained from (Levin and Reeds 1977).) Using the identity in Lemma 2, or, equivalently, by exploiting known properties of the digamma and trigamma functions, we have

$$\frac{\partial h_{\vec{n}_j}}{\partial \alpha} = -\sum_{\ell=0}^{n_j-1} \frac{1}{\ell+\alpha} + \sum_{k=1}^{q(i)} \sum_{\ell=0}^{n_{jk}-1} \frac{1}{\ell q(i)+\alpha}, \text{ and}$$

$$\frac{\partial^2 h_{\vec{n}_j}}{\partial \alpha^2} = \sum_{\ell=0}^{n_j-1} \frac{1}{(\ell+\alpha)^2} - \sum_{k=1}^{q(i)} \sum_{\ell=0}^{n_{jk}-1} \frac{1}{(\ell q(i)+\alpha)^2} \ .$$

It suffices to show that $\frac{\partial^2 h_{\vec{n}_j}}{\partial \alpha^2}$ is always negative under the conditions of the theorem. If there are at least two $n_{jk} > 0$, then

$$\frac{\partial^2 h_{\vec{n}_j}}{\partial \alpha^2} \leq \sum_{\ell=0}^{n_j-1} \frac{1}{(\ell+\alpha)^2} - \frac{2}{\alpha^2}$$

simply by ignoring all those negative terms with $\ell \geq 1$. Now we approximate it by the infinite sum of quadratic reciprocals:

$$\begin{aligned}
\frac{\partial^2 h_{\vec{n}_j}}{\partial \alpha^2} &\leq \sum_{\ell=0}^{n_j-1} \frac{1}{(\ell+\alpha)^2} - \frac{2}{\alpha^2} \\
&= -\frac{1}{\alpha^2} + \frac{1}{(1+\alpha)^2} + \sum_{\ell=2}^{n_j-1} \frac{1}{(\ell+\alpha)^2} \\
&< -\frac{1}{\alpha^2} + \frac{1}{(1+\alpha)^2} + \sum_{\ell=2}^{\infty} \frac{1}{\ell^2} \\
&= -\frac{1}{\alpha^2} + \frac{1}{(1+\alpha)^2} + \frac{\pi^2}{6} - 1 \, ,
\end{aligned}$$

which is negative for any $\alpha \leq 1$ (the gap between the two fractions containing α obviously decreases with the increase of α, so it is enough to check the sign for the largest value $\alpha = 1$). Thus we have $\frac{\partial^2 h_{\vec{n}_j}}{\partial \alpha^2} < 0$. □

The concavity of $h_{\vec{n}_j}$ is useful for the following reason.

Lemma 9
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ *and* $j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}$ *such that* $\nexists k : n_{jk} = n_j$. *If* $\alpha \leq q(\mathbf{S})$ *and* $\frac{\partial h_{\vec{n}_j}}{\partial \alpha}$ *is non-negative, then*

$$h_{\vec{n}_j}(\alpha/q(\mathbf{T})) \leq h_{\vec{n}_j}(\alpha/q(\mathbf{S})) \textit{ for every } \mathbf{T} \supseteq \mathbf{S}.$$

Proof. Since $\nexists k : n_{jk} = n_j$ and $\alpha/q(\mathbf{S}) \leq 1$, we have that $h_{\vec{n}_j}$ is concave (Lemma 8) and since $\frac{\partial h_{\vec{n}_j}}{\partial \alpha} \geq 0$, $h_{\vec{n}_j}$ is non-decreasing. □

The final step to improve the upper bound is to consider any local score of a parent set $\mathbf{S}$ as a function of the (log-)probabilities over full mass functions.

Lemma 10
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$ *and* $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$. *Then*

$$\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) \leq \sum_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} :\, j \neq j^\star} \mathit{ML}(\vec{n}_j) + \log \Pr(\mathcal{D}_{\vec{n}_{j^\star}} | \vec{\alpha}_{j_{\mathbf{S}}}),$$

where $j^\star = \arg\min_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}} \log \Pr(\mathcal{D}_{\vec{n}_j} | \vec{\alpha}_{j_{\mathbf{S}}})$.

Proof. We rewrite $n_{j_{\mathbf{S}},k}$ as the sum of counts from full mass functions: $n_{j_{\mathbf{S}},k} = \sum_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} :\, j^{\mathbf{S}} = j_S} n_{jk}$. Thus, $\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}})$ is the log probability $\log \Pr(\mathcal{D}_{\vec{n}_{j_{\mathbf{S}}}} | \vec{\alpha}_{j_{\mathbf{S}}})$ of observing a data sequence with counts $\vec{n}_{j_{\mathbf{S}}} = (\sum_{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} :\, j^{\mathbf{S}} = j_{\mathbf{S}}} n_{jk})_{k \in \mathcal{X}_i}$ under the Dirichlet-multinomial with parameter vector $\vec{\alpha}_{j_{\mathbf{S}}}$. Assume an arbitrary order for the full mass functions related to elements in $\{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} : j^{\mathbf{S}} = j_{\mathbf{S}}\}$ and name them $j_1, \ldots, j_w$, with $w = |\{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}} : j^{\mathbf{S}} = j_{\mathbf{S}}\}|$. Exploiting the conjugacy multinomial-Dirichlet we can express this probability as a product of conditional probabilities:

$$\Pr(\mathcal{D}_{\vec{n}_{j_{\mathbf{S}}}} | \vec{\alpha}_{j_{\mathbf{S}}}) = \prod_{\ell=1}^{w} \Pr\left(\mathcal{D}_{\vec{n}_{j_\ell}} \middle| \sum_{t=1}^{\ell-1} \vec{n}_{j_t} + \vec{\alpha}_{j_{\mathbf{S}}} \right),$$

$$\begin{aligned}\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) &= \sum_{\ell=1}^{w} \log \Pr\left(\mathcal{D}_{\vec{n}_{j_\ell}} \middle| \sum_{t=1}^{\ell-1} \vec{n}_{j_t} + \vec{\alpha}_{j_{\mathbf{S}}}\right) \\ &\leq \log \Pr(\vec{n}_{j_1} | \vec{\alpha}_{j_{\mathbf{S}}}) + \sum_{t=2}^{w} \mathrm{ML}(\vec{n}_{j_t}).\end{aligned}$$

These are obtained by applying expression 3.3 to all but the first term. Since the order is arbitrary, we can pick one in our best interest and the result follows. □

While the bound of Lemma 10 is valid for $\mathbf{S}$, it gives no assurances about its supersets $\mathbf{T}$, so it is of little direct use (if we need to compute it for every $\mathbf{T} \supset \mathbf{S}$, then it is better to compute the scores themselves). To address that, we replace the first term of the right-hand side summation with a proper upper bound. The maximum likelihood terms are already valid terms, as discussed earlier.

We note that Theorem 4 is in fact much simpler than its formal enunciation. Unfortunately, this is unavoidable, since we are combining different possible bounds for the term $\log \Pr(\mathcal{D}_{\vec{n}_{j^\star}} | \vec{\alpha}_{j_{\mathbf{S}}})$ that appears in Lemma 10 into one bound, while also keeping all the other maximum likelihood bounds. Moreover, to make Theorem 4 slightly more compact, we sum all maximum likelihood (ML) terms (first summation in the expression) and then we discard one of them (the first negative ML term) in order to (potentially) replace it with a better bound. This is the only reason why the definition of $\underline{h}$ in the following theorem looks unpleasant to the eyes.

Theorem 4 (ub_h)
Let $\mathbf{S} \subseteq \mathbf{X}^{\setminus i}$, $\alpha = \alpha_{ess}/q(\mathbf{S})$, $j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}$, *and* $\overline{h}_{\vec{n}_j}(\alpha) = h_{\vec{n}_j}(\alpha)$ *if* $\alpha \leq 1$ *and* $\frac{\partial h_{\vec{n}_j}}{\partial \alpha} \geq 0$, *and zero otherwise. Let*

$$\begin{aligned}\underline{h}(\mathbf{S}, j_{\mathbf{S}}) = \sum_{\substack{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}: \\ j^{\mathbf{S}} = j_{\mathbf{S}}}} ML(\vec{n}_j) + \min_{\substack{j \in \mathcal{D}_u^{\mathbf{X}^{\setminus i}}: \\ j^{\mathbf{S}} = j_{\mathbf{S}}}} \Big(- ML(\vec{n}_j) \\ + \min\{ML(\vec{n}_j);\ f(\mathbf{X}^{\setminus i}, j) + g(\mathbf{X}^{\setminus i}, j, \alpha);\ \overline{h}_{\vec{n}_j}(\alpha)\}\Big).\end{aligned}$$

Then $\mathrm{LLBDeu}(\mathbf{S}, j_{\mathbf{S}}) \leq \underline{h}(\mathbf{S}, j_{\mathbf{S}})$. *Moreover, if* $\mathrm{LBDeu}(\mathbf{S}') \geq \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{h}(\mathbf{S}, j_{\mathbf{S}}) = \underline{h}(\mathbf{S})$ *for some* $\mathbf{S}' \subset \mathbf{S}$, *then* $\mathbf{S}$ *and all its supersets are not in* $\mathcal{L}_i$.

Proof. For parent set $\mathbf{S}$, the bound based on $\mathrm{ML}(\vec{n}_j)$ only (first option in the inner minimisation, which cancels out the double ML terms) is valid by Corollary 2. The other two options rely on Lemma 10 and their own results: the

bound on $f(\mathbf{X}^{\setminus i}, j) + g(\mathbf{X}^{\setminus i}, j, \alpha)$ is valid by Lemma 6, while the bound based on $\overline{h}_{\vec{n}_j}(\alpha)$ comes from Lemma 9, and thus the result holds for $\mathbf{S}$. Take $\mathbf{T} \supset \mathbf{S}$. It is straightforward that

$$\mathrm{LBDeu}(\mathbf{T}) \leq \sum_{j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}}} \underline{h}(\mathbf{T}, j_{\mathbf{T}}) =$$

$$\sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \left(\sum_{j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}} \colon j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}} \underline{h}(\mathbf{T}, j_{\mathbf{T}}) \right) \leq \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{h}(\mathbf{S}, j_{\mathbf{S}}),$$

since $\sum_{j_{\mathbf{T}} \in \mathcal{D}_u^{\mathbf{T}} \colon j_{\mathbf{T}}^{\mathbf{S}} = j_{\mathbf{S}}} \underline{h}(\mathbf{T}, j_{\mathbf{T}}) \leq \underline{h}(\mathbf{S}, j_{\mathbf{S}})$, because both sides run over the same full instantiations and the right-hand side use the tighter minimisation of Expression equation 4 only once, while the left-hand side can use that tighter minimisation once every $j_{\mathbf{T}}$, and Lemmas 6 and 9 ensure that the computed values $f(\mathbf{X}^{\setminus i}, j) + g(\mathbf{X}^{\setminus i}, j, \alpha)$ and $\overline{h}_{\vec{n}_j}(\alpha)$ are valid for $\mathbf{T}$. □

As with previous theorems, Theorem 4 gives us a new upper bound on the local score of a parent set $\mathbf{S}$

$$\mathrm{ub}_h(\mathbf{S}) = \underline{h}(\mathbf{S}) = \sum_{j_{\mathbf{S}} \in \mathcal{D}_u^{\mathbf{S}}} \underline{h}(\mathbf{S}, j_{\mathbf{S}}) \geq \max_{\mathbf{T} : \mathbf{T} \supset \mathbf{S}} \mathrm{LBDeu}(\mathbf{T}).$$

3.6 Combining the Bounds

We note that bound ub_g of the previous section was obtained in a similar way as ub_f, and we prove that $\mathrm{ub}_g(\mathbf{S}) \leq \mathrm{ub}_f(\mathbf{S})$ for any candidate parent set $\mathbf{S}$. Conversely, ub_h bears no such relation to ub_f as we derived it through a new route, studying the properties of the likelihood function. This is to our advantage, as due to their independent theoretical derivations, ub_g and ub_h prune different regions of the search space and can be effectively combined into a tighter bound $\mathrm{ub}_{g,h} = \min\{\mathrm{ub}_g; \mathrm{ub}_h\}$.

3.7 Experiments

This work focuses on new theoretical derivations leading to tighter bounds, and thus an empirical analysis is beyond its scope. Nonetheless, we illustrate possible gains as well as a comparison of the different bounds in simple benchmark

datasets in Figures 3.2 and 3.3. The code for computing these bounds and reproducing the experiments is available at `https://github.com/alcorreia/bdeu-structure-learning`.

For small datasets, it is feasible to score every candidate parent set so that we can compare how far the upper bounds for a given parent set S (and all its supersets) are from the true best score among itself and its supersets. Figure 3.2 shows such a comparison for variable *Standard-of-living-index* in the *CMC* dataset (Dua and Graff 2017), which has 10 variables and 1,473 instances. It is clear that the new bound $\text{ub}_{g,h}$ is much tighter than the current best bound in the literature (here called ub_f) and improves considerably towards the true best score (only available because this particular dataset is not too large).

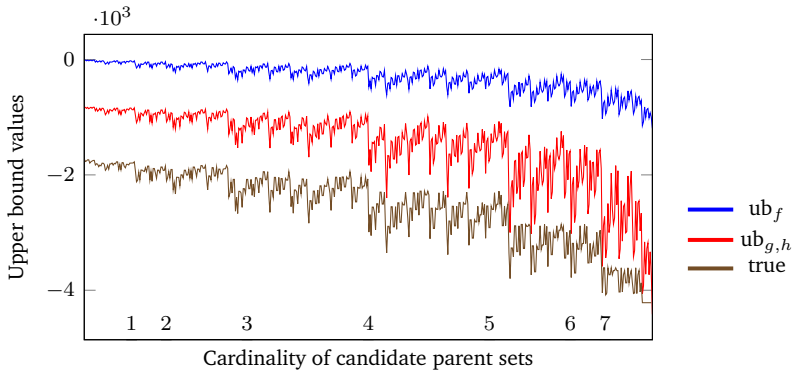

Figure 3.2: Upper bound values for each candidate parent set for variable Standard-of-living-index in the *CMC* dataset (Dua and Graff 2017). Parent sets are arbitrarily ordered within each cardinality (neighbourhood in the graph within same cardinality is not relevant).

For larger datasets (more than 10 variables), evaluating all candidate parent sets becomes computationally impracticable, so instead we evaluate the number of scores computed with each bound. In Figure 3.3, we see the new bounds considerably reduced the number of scores computed, which translates into smaller lists of potentially optimal parent sets L_i (see Definition 2). This goes to show the practical value of tighter upper bounds, as we save computing time in both steps of BNSL: parent set identification (fewer scores to compute) and structure optimisation (smaller search space). We can also observe in Figure 3.3 that, in these four datasets, our new bounds have a large advantage over ub_f only for relatively large maximum number of parents. That might seem restrictive

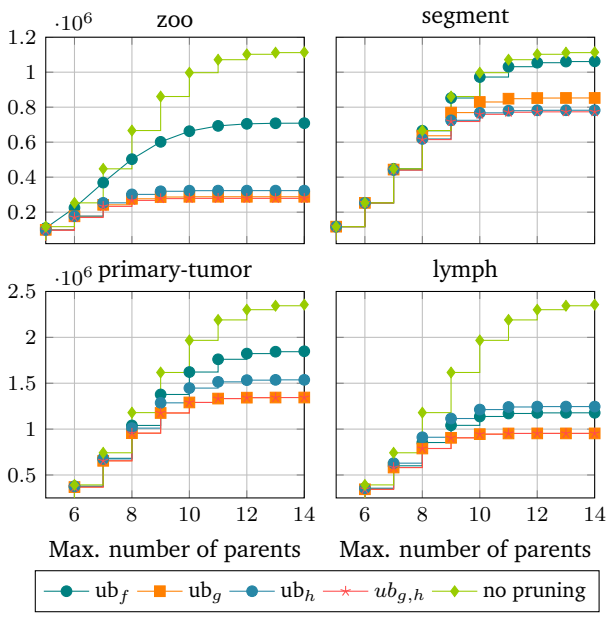

Figure 3.3: Plot of the number of scores computed per maximum number of parents with different pruning bounds for four UCI datasets (Dua and Graff 2017) with 17 (zoo, segment) and 18 (primary-tumor, lymph) variables. The scores were computed using breadth-first search.

since we are probably fine using ub_f for small numbers of parent sets. However, exact structure learning with large maximum number of parents is exactly the scenario where we need tight bounds the most, and in that case our bounds might prove extremely useful. We believe these new upper bounds ub_g and ub_h are an important step towards making exact structure learning without limits on the number of parents more efficient and scalable.

We report similar results for a total of 22 UCI datasets (Dua and Graff 2017) in Tables 3.2 and 3.3[2], where we show the number of parent sets pruned for different maximum in-degrees (maximum number of parents per node) with each of the four bounds: the previously proposed ub_f (de Campos and Ji 2011; Cussens and Bartlett 2015), and our bounds ub_g, ub_h and $\text{ub}_{g,h}$. We see from

[2]We split the tables only to fit the limited space in a page. Both tables contain the same information but regarding different datasets.

Table 3.2: Number of computations pruned ($|L^c| = |\text{search space}| - |L|$) with each bound: ub_f, ub_g, ub_h and $\text{ub}_{g,h}$. Each dataset is characterised by its number of variables and observations, n and N, and the number of all possible parent combinations |search space|. The maximum imposed in-degree is denoted in-d and $\frac{|\text{ub}_g < \text{ub}_h|}{|\text{ub}_h < \text{ub}_g|}$ is the proportion of times ub_g was tighter than ub_h.

Dataset	n	N	|search space|	in-d	$\|L^c_f\|$	$\|L^c_g\|$	$\|L^c_h\|$	$\|L^c_{g,h}\|$	$\frac{\|\text{ub}_g<\text{ub}_h\|}{\|\text{ub}_h<\text{ub}_g\|}$
car	7	1,728	448	5	0	36	129	129	1.361
				6	6	43	135	136	1.661
				∞	6	43	135	136	1.661
glass	8	214	1,024	5	0	11	9	12	1.488
				6	0	44	40	45	1.483
				7	0	52	48	53	1.483
diabetes	9	768	2,304	5	0	0	0	0	∞
				7	0	72	184	184	123.176
				8	0	81	193	193	123.176
nursery	9	12,960	2,304	5	0	0	342	342	6.176
				7	8	188	626	630	5.331
				8	16	197	635	639	5.331
breast-cancer	10	286	5,120	5	166	600	227	600	$4.475 \cdot 10^{-2}$
				7	589	1,603	1,019	1,603	$4.206 \cdot 10^{-2}$
				9	660	1,703	1,119	1,703	$4.206 \cdot 10^{-2}$
tic-tac-toe	10	958	5,120	5	0	0	0	0	0
				7	2	2	0	2	0
				9	101	101	26	101	0
cmc	10	1,473	5,120	5	0	23	35	47	7.556
				7	6	746	766	828	6.045
				9	16	846	866	928	6.045
heart-h	12	294	24,576	5	0	252	86	252	1.434
				8	1,109	7,469	6,090	7,504	1.019
				11	1,321	8,273	6,894	8,308	1.019
solar-flare	12	1,066	24,576	5	884	1,810	2,170	2,462	2.799
				8	3,741	8,890	10,561	11,043	2.885
				11	4,043	9,672	11,348	11,834	2.877
vowel	14	990	$1.147 \cdot 10^5$	5	1,564	1,833	1,707	1,837	0.182
				9	7,579	22,701	21,962	22,729	$6.152 \cdot 10^{-2}$
				13	8,350	26,298	25,411	26,330	$6.162 \cdot 10^{-2}$
zoo	17	101	$1.114 \cdot 10^6$	5	7,760	18,026	18,818	20,604	0.252
				11	$3.782 \cdot 10^5$	$7.834 \cdot 10^5$	$7.483 \cdot 10^5$	$7.925 \cdot 10^5$	0.104
				16	$4.054 \cdot 10^5$	$8.262 \cdot 10^5$	$7.91 \cdot 10^5$	$8.353 \cdot 10^5$	0.104
vote	17	435	$1.114 \cdot 10^6$	5	0	0	0	0	0.919
				11	40,544	$2.594 \cdot 10^5$	$2.126 \cdot 10^5$	$2.776 \cdot 10^5$	0.414
				16	55,067	$3.021 \cdot 10^5$	$2.552 \cdot 10^5$	$3.203 \cdot 10^5$	0.414

the results that bound ub_g always leads to more aggressive pruning than ub_f, which is as expected since the former is provenly tighter. These experiments also show there is no clear winner between ub_g and ub_h, since the effectiveness of the upper bounds seems to be heavily dependent on the dataset.

Finally, we point out that the mathematical results may seem harder to apply than they actually are. Computing $\text{ub}_g(S)$ and $\text{ub}_h(S)$ to prune a parent set S and all its supersets can be done in linear time, as one pass through the data is enough to collect and process all required counts; more sophisticated data

Table 3.3: Number of computations pruned ($|L^c| = |\text{search space}| - |L|$) with each bound: ub_f, ub_g, ub_h and $\text{ub}_{g,h}$. Each dataset is characterised by its number of variables and observations, n and N, and the number of all possible parent combinations |search space|. The maximum imposed in-degree is denoted in-d and $\frac{|\text{ub}_g < \text{ub}_h|}{|\text{ub}_h < \text{ub}_g|}$ is the proportion of times ub_g was tighter than ub_h.

Dataset	n	N	|search space|	in-d	$\|L^c_f\|$	$\|L^c_g\|$	$\|L^c_h\|$	$\|L^c_{g,h}\|$	$\frac{\|\text{ub}_g<\text{ub}_h\|}{\|\text{ub}_h<\text{ub}_g\|}$
segment	17	2,310	$1.114 \cdot 10^6$	5	0	0	0	0	0.184
				11	39,948	$2.229 \cdot 10^5$	$2.915 \cdot 10^5$	$2.915 \cdot 10^5$	0.256
				∞	51,902	$2.614 \cdot 10^5$	$3.317 \cdot 10^5$	$3.317 \cdot 10^5$	0.256
pendigits	17	10,992	$9.83 \cdot 10^5$	5	0	0	0	0	0
				11	0	2,386	47,757	41,619	$4.143 \cdot 10^{-2}$
				16	0	17,445	76,982	70,321	$4.383 \cdot 10^{-2}$
glass	8	214	1,024	5	0	11	9	12	1.488
				6	0	44	40	45	1.483
				7	0	52	48	53	1.483
lymph	18	148	$2.359 \cdot 10^6$	5	7,295	8,344	6,076	8,344	12,375.846
				11	$1.02 \cdot 10^6$	$1.237 \cdot 10^6$	$9.489 \cdot 10^5$	$1.237 \cdot 10^6$	73,280.769
				17	$1.182 \cdot 10^6$	$1.406 \cdot 10^6$	$1.114 \cdot 10^6$	$1.406 \cdot 10^6$	73,327.538
primary-tumor	18	339	$2.359 \cdot 10^6$	5	2,460	3,555	2,667	3,555	$5.465 \cdot 10^{-2}$
				11	$4.292 \cdot 10^5$	$8.572 \cdot 10^5$	$6.751 \cdot 10^5$	$8.572 \cdot 10^5$	$6.856 \cdot 10^{-3}$
				17	$5.105 \cdot 10^5$	$1.015 \cdot 10^6$	$8.223 \cdot 10^5$	$1.015 \cdot 10^6$	$6.797 \cdot 10^{-3}$
vehicle	19	846	$4.981 \cdot 10^6$	5	0	108	54	108	1.197
				12	$6.614 \cdot 10^5$	$2.082 \cdot 10^6$	$1.848 \cdot 10^6$	$2.12 \cdot 10^6$	0.474
				18	$7.582 \cdot 10^5$	$2.319 \cdot 10^6$	$2.084 \cdot 10^6$	$2.358 \cdot 10^6$	0.473
hepatitis	20	155	$7.864 \cdot 10^6$	5	0	0	0	0	397.196
				12	$2.155 \cdot 10^6$	$3.341 \cdot 10^6$	$2.338 \cdot 10^6$	$3.341 \cdot 10^6$	8,198.164
				19	$2.599 \cdot 10^6$	$3.795 \cdot 10^6$	$2.905 \cdot 10^6$	$3.795 \cdot 10^6$	6,100.199
colic	23	368	$9.647 \cdot 10^7$	5	1,170.125	2,415	934.375	2,415	∞
				14	$2.116 \cdot 10^7$	$2.122 \cdot 10^7$	$2.042 \cdot 10^7$	$2.122 \cdot 10^7$	∞
				22	$2.277 \cdot 10^7$	$2.284 \cdot 10^7$	$2.203 \cdot 10^7$	$2.284 \cdot 10^7$	∞
autos	26	205	$8.724 \cdot 10^8$	5	$1.388 \cdot 10^5$	$1.829 \cdot 10^5$	$1.544 \cdot 10^5$	$1.829 \cdot 10^5$	∞
				15	$1.265 \cdot 10^8$	$1.265 \cdot 10^8$	$1.258 \cdot 10^8$	$1.265 \cdot 10^8$	45,904.333
				25	$1.432 \cdot 10^8$	$1.432 \cdot 10^8$	$1.425 \cdot 10^8$	$1.432 \cdot 10^8$	45,904.333
flags	29	194	$7.785 \cdot 10^9$	5	$2.782 \cdot 10^5$	$2.834 \cdot 10^5$	$1.275 \cdot 10^5$	$2.834 \cdot 10^5$	∞
				17	$1.085 \cdot 10^9$	$1.085 \cdot 10^9$	$1.083 \cdot 10^9$	$1.085 \cdot 10^9$	∞
				28	$1.196 \cdot 10^9$	$1.196 \cdot 10^9$	$1.194 \cdot 10^9$	$1.196 \cdot 10^9$	∞

structures, such as AD-trees (Moore and Lee 1998), might allow for even greater speedups. Since calculating a score already takes linear time in the number of data samples, we have cheap bounds which are provably superior to the current state-of-the-art pruning for BDeu.

3.8 Structure Learning and Tractability

The work presented in this chapter is also a push for exact inference algorithms, as the rest of the research in this thesis. In the case of Bayesian Networks, we promote exact structure learning—also a form of inference—rather than exact

probabilistic queries as with Probabilistic Circuits. However, our new upper bounds might also prove useful in the overall context of tractable probabilistic models. They are readily applicable to a number of algorithms that constrain structure learning to Bayesian Networks supporting tractable inference routines (Elidan and Gould 2008; Nie et al. 2014, 2017; Scanagatta et al. 2015, 2016, 2018; Benjumeda et al. 2016, 2019).

As we have seen in Chapter 2, most score-based structure learning algorithms penalise the number of parameters in the Bayesian Network in an effort to avoid overfitting. Unfortunately, a compact model does not guarantee tractable inference, and we need other ways to favour tractable models in our structure search. Most methods use tree-width as a measure of tractability, since the worst-case inference complexity is exponential in the tree-width of the Bayesian Network structure (Kwisthout et al. 2010). These methods navigate the search space and discard or do not consider structures with tree-width above certain threshold (Elidan and Gould 2008; Nie et al. 2014, 2017; Scanagatta et al. 2015, 2016, 2018). The tree-width can also be expressed as the width of the optimal elimination order of a graph, and a few methods exploit this quantity in their search for tractable BNs (Benjumeda et al. 2016, 2019).

All these methods use some form scoring function to navigate the search space and compare the goodness of fit of each graph. Therefore, our new upper bounds might be instrumental in speeding up learning of tractable Bayesian Networks as well. In fact, they might be particularly useful in this context since learning tractable BNs requires an extra expensive computational step—determining the tree-width (or optimal elimination order) of a graph is an NP-hard problem in and of itself (Arnborg et al. 1987)—and effectively pruning the search space becomes even more essential.

3.9 Conclusion

We introduced new theoretical upper bounds for exact structure learning of Bayesian networks with the BDeu score by studying the score function from multiple angles. These bounds are provably tighter than previous results and shall provide significant benefits in reducing the search space in candidate parent set identification in BNSL and potentially other applications involving independence assumptions. Moreover, the two new bounds we proposed are derived via two different perspectives on the BDeu score: we studied the gamma function to arrive at ub_g and exploited the likelihood function to get to ub_h. That gives us two distinct tools to prune the search space that, as shown in our exper-

iments, have different degrees of effectiveness depending on the dataset, and thus might complement each other well.

A natural step for future research is the integration of our bounds with more sophisticated data structures and search algorithms. As an example, branch-and-bound methods are particularly promising as they not only consider the parent sets and its corresponding full instantiations but also partial instantiations that are formed by disallowing some variables to be parents in some of the branches. Our results also open new routes for further theoretical work in exact structure learning. Notably, we conjecture that the maximum likelihood estimation terms still leave room for tighter bounds.

Chapter 4

An Experimental Study of Prior Dependence in Bayesian Network Structure Learning

The Bayesian Dirichlet equivalent uniform (BDeu) function is a popular score to evaluate the goodness of a Bayesian network structure given complete categorical data. Despite its interesting properties, such as likelihood equivalence, it does require a prior expressed via a user-defined parameter known as Equivalent Sample Size (ESS), which significantly affects the final structure. We study conditions to obtain prior independence in BDeu-based structure learning. We show in experiments that the amount of data needed to render the learning robust to different ESS values is prohibitively large, even in big data times.

This chapter is almost integrally based on **Alvaro Correia**, Cassio de Campos and Linda C. van der Gaag: An Experimental Study of Prior Dependence in Bayesian Network Structure Learning, *International Symposium on Imprecise Probabilities: Theories and Applications (ISIPTA)*, 2019.

4.1 Introduction

In this short chapter, we continue to study the Bayesian Dirichlet equivalent uniform (BDeu) score, but this time with a focus on the prior expressed via the equivalent sample size (ESS) and its influence on the final learnt structure. As we demonstrate in the experiments, the choice of ESS remains consequential even for extremely large sample sizes, making it virtually impossible to break prior dependence in structure learning with the BDeu score.

Bayesian networks are a class of probabilistic graphical models based on a Directed Acyclic Graph (DAG) $\mathcal{G}$ that defines a factorisation of the joint probability distribution over a set of random variables $\mathbf{X} = \{X_1, ..., X_m\}$. One can learn the DAG (also called structure or graph) $\mathcal{G}$ from complete discrete data $\mathcal{D}$ via the popular BDeu score function (Buntine 1991; Cooper and Herskovits 1992; Heckerman et al. 1995), which aims at finding a *maximum a posteriori* (MAP) $\mathcal{G}$ that maximises $P(\mathcal{G}|\mathcal{D})$ (under a uniform prior over $\mathcal{G}$). See Chapter 2 for an overview of Bayesian networks and scoring functions like the BDeu.

For the sake of completeness we briefly restate the BDeu score here, making explicit the dependence on the ESS α. The BDeu score for a graph $\mathcal{G}$ is defined by the marginal likelihood of the data $\mathcal{D}$ given $\mathcal{G}$ and the ESS $\alpha > 0$:

$$\text{BDeu}(\mathcal{G}, \alpha) = \sum_{i=1}^{m} \sum_{j=1}^{q(\text{pa}(X_i))} \left[\log \left(\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \right) + \sum_{k=1}^{q(X_i)} \log \left(\frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \right) \right].$$

As evident in the definition above, the BDeu score decomposes into a sum of m terms, each depending exclusively on a single variable X_i and its parents $\text{pa}(X_i)$ in graph $\mathcal{G}$. We use $q(\cdot)$ to denote the size of the state space of a set of variables, with $q(i)$ the arity of X_i and $q(\text{pa}(X_i))$ the number of joint instantiations of its parents. We use n_{ijk} to represent the number of observations with $X_i = k$ and $\text{pa}(X_i) = j$, and $n_{ij} = \sum_k n_{ijk}$. Finally, we define the pseudo-counts as

$$\alpha_{ijk} = \frac{\alpha}{q(i)q(\text{pa}(X_i))} \text{ and } \alpha_{ij} = \frac{\alpha}{q(i)}.$$

Structure learning with BDeu requires the definition of the Dirichlet parameters, which is done through $\alpha > 0$, roughly expressing the strength of our prior belief. However, there is no consensus on what value an 'uninformative' α

should take and several studies have focused on measuring the influence of α on the final structure (Scutari 2016, 2018; Silander et al. 2007), analysing the asymptotic behaviour of the BDeu for $\alpha \to 0$ and $\alpha \to \infty$ (Steck 2008; Steck and Jaakkola 2003; Ueno 2010, 2011), or finding the optimal α (Silander et al. 2007; Steck 2008).

Nonetheless, to the best of our knowledge, no work has directly addressed the robustness of BDeu-based structure learning to variations of the ESS. By robustness we mean prior-independence, i.e. for large enough data, one should expect the structure learning algorithm to produce the same network regardless of the prior knowledge expressed via the ESS. As we show in the experiments, even for a small number of variables the amount of data required to achieve such robustness is prohibitively large. That suggests the prior on the BDeu function might be too strong for some real-world applications, where other scores (or some variation of the BDeu score) might be more adequate.

4.2 Experiments

We conducted experiments with three known Bayesian networks (Beinlich et al. 1989; Sachs et al. 2005; Spiegelhalter et al. 1993) and 16 datasets from the UCI Machine Learning repository (Dua and Graff 2017) to study the influence of the ESS. In all experiments, we assumed complete categorical data (we discretised continuous variables into two categories by their median values, when needed). As we wanted to study the intrinsic behaviour of BDeu-based structure learning, and not the particularities of a given approximate solver, we focused on exact solutions. For that there are a number of exact solvers available (de Campos and Ji 2011; Silander and Myllymäki 2006; Yuan and Malone 2012), and we used GOBNILP (Barlett and Cussens 2013), which finds the optimal graph via integer linear programming.

In the experiments in Figure 4.1 and 4.2, we assumed a given ordering of the variables, i.e. for any nodes (variables) X and Y, if X precedes Y in the ordering, then an arc between X and Y (if it exists) must be directed from X to Y. That restriction considerably reduces the search space and allows us to consider larger sample sizes, while still guaranteeing an exact solution. Conversely, the UCI datasets are small enough that we could gather exact results both with and without order constraints.

4.2.1 Graph Complexity

The ESS can be interpreted as a regularizer on the structure of the Bayesian network. Hence, we start by investigating how it affects the total number of arcs (parents) in the network. In particular, we are interested in the interplay between the *sample size* n and the ESS in determining the complexity of the network.

We sampled data from three known networks, Sachs (Sachs et al. 2005) with $m = 11$ variables, Child (Spiegelhalter et al. 1993) with $m = 20$ variables, and Alarm (Beinlich et al. 1989) with $m = 37$ variables. Subsequently we learned the structure with GOBNILP while varying the sample size n and the ESS. We repeated that process for 30 different orderings and reported the average number of arcs (across orderings) in Figure 4.1. The results indicate the graph increases in complexity with the ESS. Indeed, the number of arcs is expected to grow *almost* monotonically to the maximum (complete graph) for large values of α (Silander and Myllymäki 2006; Steck 2008; Steck and Jaakkola 2003; Ueno 2010, 2011).

A more interesting analysis that received little attention in the literature is how the complexity of the graph varies with the sample size. Naturally, one would expect that, for large datasets, the prior would have little effect on the

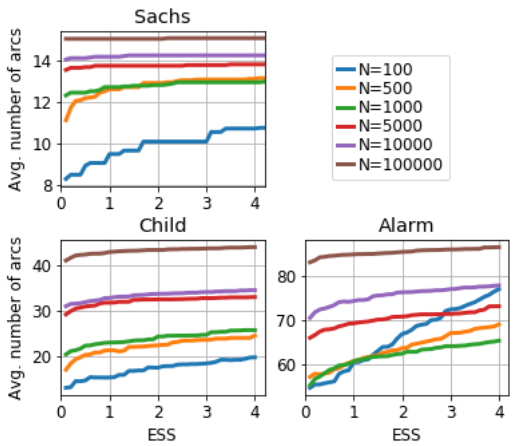

Figure 4.1: Average number of arcs as a function of the ESS for different sample sizes n. We denote n in uppercase N for ease of reading. Scale of the graphs varies.

learned network. That is what we observe for the Sachs network in Figure 4.1: the number of arcs remains constant across all ESS values for $n \geq 10^5$. However, considering Sachs contains only 11 variables, that is an extremely high number of data points to guarantee robustness over a relatively small range of ESS values. For the other networks, no amount of data ensured prior-independence. The number of arcs increased with the ESS, and providing more data points did not alleviate this trend significantly.

In a statistical sense, the ESS is not a typical Dirichlet prior because it also defines the number of parameters in the model. It expresses a trade-off between regularisation and complexity (Steck and Jaakkola 2003), which increases with the ESS and with sample size n, as shown in the experiments. That trade-off partially explains why it is hard to avoid prior-dependence in BDeu-based structure learning.

4.2.2 Robustness

To study prior-independence in BDeu-based structure learning, one needs a metric that captures the influence of the ESS on the final structure.

Definition 3 (Robust Interval)
We define robust interval (RI) as the largest range of ESS values for which all obtained optimal structures (for each ESS) are Markov equivalent.

$$RI := \underset{[\alpha_1,\alpha_2]}{\arg\max}\{|\alpha_2 - \alpha_1| \colon G^*(\alpha') \equiv G^*(\alpha'), \forall \alpha', \alpha' \in [\alpha_1, \alpha_2]\},$$

where $G^*(\alpha) = \arg\max_G BDeu(G, \alpha)$ is the optimal graph for a given ESS, and we use $\equiv$ to denote Markov equivalence.

Intuitively, the larger the robust interval (or RI), the more prior-independent the learning algorithm (for a given dataset). We do not distinguish structures representing the same set of conditional independence statements (commonly referred to as Markov equivalent), as they encode the same 'information' and have the same BDeu score (Heckerman et al. 1995)[1]. The advantage of the RI against other metrics, such as structural Hamming distance (SHD) (Tsamardinos et al. 2006), is that it does not require a gold standard network and also signals a safe range of ESS values over which the influence of the prior is mitigated.

[1] See Chapter 2 for a discussion on score equivalence.

Note that the RI is only meaningful if reported for non-complete graphs. For $\alpha \to \infty$, the learned structure tends to a complete graph (Ueno 2010), and it follows that, for large enough α, the structure is 'infinitely' robust but overfitted, which is an uninteresting result. In the experiments, we computed the RI by finding the optimal structure with α covering the range $(0.1, 4.0)$ in increments of 0.1. By using increments we do not obtain the true RI but a conservative estimate: the true RI is either smaller (due to unobserved variations in-between increments) or at most 0.2 larger.

We report the results in Figure 4.2, where for each dataset and for each pair (α, n), we see the average RI of 30 randomly sampled orderings.

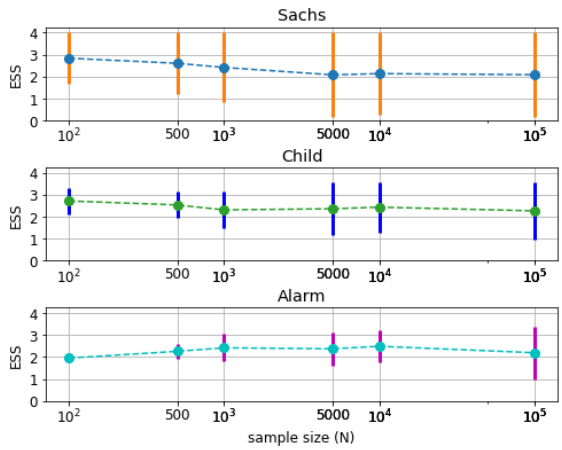

Figure 4.2: Robust Interval for Sachs (Sachs et al. 2005), Child (Spiegelhalter et al. 1993) and Alarm (Beinlich et al. 1989) as a function of the sample size n, which again we denote in uppercase N for ease of reading.

In a Bayesian framework, we want the prior to become less relevant in determining the final structure of the network as we gather more data. This in turn should result in larger robust intervals. In Figure 4.2, we see the BDeu does comply with that requirement to some extent, as the robust interval does increase with n. However, the amount of data required to cover the small ESS range we analysed is already prohibitively large even for a small number of variables. That supports recent studies that claim the BDeu is not fit for sparse data (Scutari 2016, 2018), but also alerts us that almost every real-world dataset might be too sparse for the BDeu.

Table 4.1: Largest ESS range yielding the same structure (RI) for UCI datasets. Scalars m and n are the number of variables and samples, respectively, RIo the average RI of 10 orderings, and RIf the RI without order constraint.

Dataset	m	n	RIo	RIf
car	7	1728	(0.1, 4.0)	(0.4, 4.0)
glass	8	214	(1.3, 2.3)	(0.3, 4.0)
spambase	8	4601	(1.2, 4.0)	(1.7, 4.0)
diabetes	9	768	(0.2, 1.7)	(1.6, 4.0)
nursery	9	12960	(1.4, 2.9)	(1.4, 4.0)
breast-cancer	10	286	(1.9, 4.0)	(2.2, 4.0)
tic-tac-toe	10	958	(1.8, 2.1)	(1.7, 2.2)
cmc	10	1473	(1.7, 2.9)	(0.8, 2.8)
heart-h	12	294	(0.8, 1.6)	(2.2, 2.9)
vowel	14	990	(0.6, 1.8)	(1.9, 4.0)
zoo	17	101	(0.6, 1.3)	(0.9, 2.1)
vote	17	435	(0.8, 1.8)	(2.3, 3.1)
segment	17	2310	(1.5, 2.9)	(2.3, 4.0)
primary-tumor	18	339	(1.1, 1.5)	(3.1, 3.5)
vehicle	19	846	(0.9, 1.7)	(3.3, 4.0)

We did the same analysis for 16 UCI datasets (Dua and Graff 2017). In these experiments, we computed both the average RI of 10 different orderings (RIo) and the RI with no constraint on the ordering (RIf). Again, in Table 4.1, we see that except for the *car* dataset, none of them had enough data to guarantee robustness of the BDeu-based structure learning with $\alpha \in (0.1, 4.0)$. Interestingly, the size of the interval did not change significantly between solutions with and without a order constraint, but the RI stabilised at slightly higher ESS values when no ordering was given.

The RI can also be seen as an indication of a safe interval at which the influence of the prior is minimal. However, for more than half of the datasets, the robust interval did not include the canonical $\alpha = 1$. That contrasts with previous studies suggesting the influence of the ESS on the learned structure is minimised when it is set to one (Ueno 2010).

All in all, the results support previous research stressing the BDeu is highly sensitive to the ESS. That is crucial when one wants to study the graph per se but may also impact the predictive power of the models. Therefore, one must be aware and accept the large influence the prior may have when using BDeu, since the amount of data will likely be insufficient to avoid prior dependence. A promising avenue for future work is to extend the analysis to parameter learning so as to investigate further the overall impact of the ESS on the learnt models.

Chapter 5

Probabilistic Circuits

Probabilistic Circuits (PCs) (Vergari et al. 2020) is a family of density representations facilitating many exact and efficient inference routines. The PC framework allow us to describe and study a diverse group of tractable probabilistic models, including Chow-Liu trees *(Chow and Liu 1968),* arithmetic circuits *(Darwiche 2003),* sum-product networks *(Poon and Domingos 2011),* cutset networks *(Rahman et al. 2014),* probabilistic sentential decision diagrams *(Kisa et al. 2014), and our own work on* generative forests *(Correia et al. 2020b). In this chapter, we give a general introduction to Probabilistic Circuits covering the most relevant literature on structure and parameter learning of PCs. The focus of this exposition will be on PCs that support tractable marginalisation and conditioning queries—commonly referred to as Sum-Product Networks (SPNs)—since that is the class of models we explore in this thesis. There are, however, a number of other interesting models supporting efficient inference routines for a larger group of probabilistic queries, including maximum a posteriori, marginal maximum a posterior and pairwise queries. We refer the reader to (Vergari et al. 2020; Choi et al. 2020) for a recent and more comprehensive survey of PCs.*

5.1 Introduction

We begin by introducing the necessary notation. To this end, let the set of variables (features) be $\mathbf{X} = \{X_1, X_2, \ldots, X_m\}$, where continuous X_i assume values in some compact set $\mathcal{X}_i \subset \mathbb{R}$ and discrete X_i assume values in $\mathcal{X}_i = \{1, \ldots, K_i\}$, where K_i is the number of states for X_i. Let the joint *feature space* of $\mathbf{X}$ be denoted as $\boldsymbol{\mathcal{X}}$. We denote joint states, i.e. elements from $\boldsymbol{\mathcal{X}}$, as $\mathbf{x}$ and let $\mathbf{x}[i]$ be the state in $\mathbf{x}$ belonging to X_i. We assume that $\mathbf{X}$ is drawn from a fixed joint distribution $\mathbb{P}^*(\mathbf{X})$ which has density $p^*(\mathbf{X})$. While the true distribution $\mathbb{P}^*$ is unknown, we assume that we have a dataset $\mathcal{D}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of n i.i.d. samples from $\mathbb{P}^*$. Similarly to previous chapters, we use $\mathcal{D}_I$ to denote a partition of $\mathcal{D}$ comprised of the instances indexed by an index set $I \subseteq \mathbb{Z}_{<n}$, and $\mathcal{D}^{\mathbf{S}}$ to represent a projection of $\mathcal{D}$ into a subset of variables $\mathbf{S} \subseteq \mathbf{X}$ as $\mathcal{D}^{\mathbf{S}}$.

When describing a directed graph $\mathcal{G}$, we refer to its set of nodes as V, reserving letters u and v for individual nodes. We denote the set of children and parents of a node v as $\mathrm{ch}(v)$ and $\mathrm{pa}(v)$, respectively. Nodes without children are referred to as *leaves*, which we shall often denote as ℓ, and nodes without parents are referred to as *roots*, which we denote as r (PCs typically have a single root). We are now ready to define a Probabilistic Circuit.

Definition 4 (Probabilistic Circuit (PC))
Given a set of random variables $\mathbf{X}$*, a PC is based on a weighted, rooted and acyclic directed graph* $\mathcal{G}$ *containing three types of nodes:* distribution nodes *defining a probability distribution over a subset of the random variables in* $\mathbf{X}$*;* sum nodes*, which compute a convex combination (a mixture) of their inputs, and* product nodes*, which compute the product of their inputs. All leaves of* $\mathcal{G}$ *are distribution nodes and all internal nodes are either sum or product nodes.*

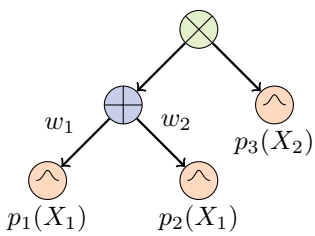

Figure 5.1: Illustration of a simple Probabilistic Circuit, with sum nodes in blue, product nodes in green, and distribution nodes in orange.

A PC is a computational graph. Much like a neural network (Goodfellow et al. 2016) a PC defines a sequence of arithmetic operations starting with the computation of probability values for each distribution node and followed by sum and product operations as specified by the graph. More precisely a PC computes a function commonly referred to as network polynomial (Darwiche 2003), which is probably best illustrated via a simple example. In Figure 5.1 we have a small PC which computes the following multilinear polynomial

$$p(\mathbf{x}) = p_3(\mathbf{x}[2])\left(w_1 p_1(\mathbf{x}[1]) + w_2 p_2(\mathbf{x}[1])\right), \tag{5.1}$$

where p_1, p_2 and p_3 are probability density (or mass) functions defined by three different distribution nodes, and w_1 and w_2 are the weights of the sum node in Figure 5.1. One of the main characteristics of PCs is that they allow us to compute the likelihood of complete and incomplete evidence with a single pass through the network, i.e. with computational cost that is linear in the size of the network (Poon and Domingos 2011; Peharz et al. 2015). Equation 5.1 makes this evident, since computing $p(\mathbf{x})$ in the PC of Figure 5.1 boils down to evaluating a multilinear polynomial of size proportional to the size of the model.

A Probabilistic Circuit is defined by its structure $\mathcal{G}$ and its parameters $\boldsymbol{\theta}$, which comprise the weights of the sum nodes as well as the parameters of each distribution node. We often refer to the density function defined by a PC simply as $p(\mathbf{x} \mid \boldsymbol{\theta}, \mathcal{G})$ to make evident the dependence on the structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$. However, when clear from the context, we shall use shorthand notation $p(\mathbf{x} \mid \boldsymbol{\theta})$ or, at times, even $p(\mathbf{x})$ as above. We often also have to refer to the densities defined by each node; as we shall see, each and every node in a PC defines a valid probability distribution. In that case, we shall denote the density defined by node v simply as $p_v(\mathbf{x})$.

Before expanding on the definition above and discussing each of the node types in detail, we must introduce a few important structural properties of PCs. These rely on the concept of scope. The *scope* of a distribution node (leaf) v, denoted $\psi(v)$, is the set of variables over which it computes a probability distribution. Given the scopes of the leaves, the scope of any internal node v (sum or product) is recursively defined as $\psi(v) = \cup_{u \in \mathrm{ch}(v)} \psi(u)$.

5.1.1 Structural Properties

The main feature of PCs is that they facilitate a wide range of *tractable* inference routines, which go hand in hand with certain structural properties, defined as follows (Darwiche 2003; Vergari et al. 2020):

1. A sum node v is called **smooth** if its children have all the same scope: $\psi(u) = \psi(u')$, for any $u, u' \in \mathrm{ch}(v)$.

2. A product node v is called **decomposable** if its children have scopes that do not overlap: $\psi(u) \cap \psi(u') = \emptyset$, for any $u, u' \in \mathrm{ch}(\pi)$, $u \neq u'$.

A PC is smooth (respectively decomposable) if all its sums (respectively products) are smooth (respectively decomposable). Smoothness and decomposability are sufficient to ensure *tractable marginalisation* in PCs. In particular, assume that we wish to evaluate the density over $\mathbf{X}_o \subset \mathbf{X}$ for evidence $\mathbf{X}_o = \mathbf{x}_o$, while marginalising $\mathbf{X}_{\neg o} = \mathbf{X} \setminus \mathbf{X}_o$. In PCs, this task reduces to performing marginalisation at the leaves (Peharz et al. 2015), that is, for each leaf v one marginalises $\psi(v) \cap \mathbf{X}_{\neg o}$, and evaluates it for the values corresponding to $\psi(v) \cap \mathbf{X}_o$. The desired marginal $p(\mathbf{x}_o)$ results from evaluating internal nodes as in computing the complete density, and hence can also be expressed as a multilinear polynomial as in Equation 5.1 and computed in time linear in the size of the model. All PCs considered in this thesis are both smooth and decomposable, and thus support exact and efficient marginalisation routines.

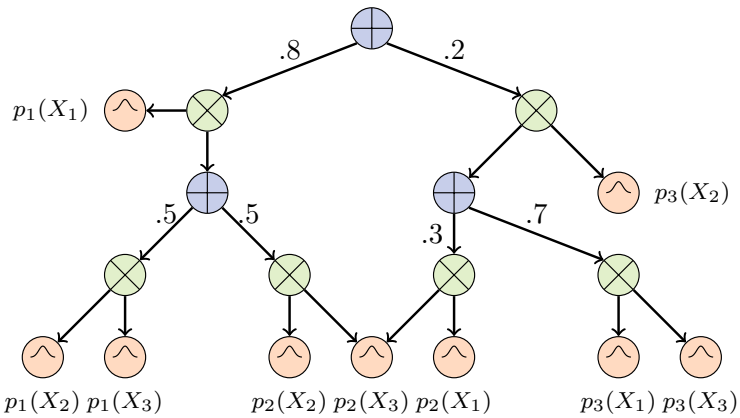

Figure 5.2: Example of a simple PC. This PC is smooth and decomposable but not necessarily deterministic. Although most of the work in PCs focus on tree structures, PCs are not constrained to be tree-shaped. In fact, the PC depicted in the figure is not a tree since $p_2(X_3)$ has more than one parent. That said, for every valid PC we can find an equivalent tree-shaped PC. In this case, it would suffice to replicate $p_2(X_3)$ so that each copy has a single parent.

Moreover, the ability to compute exact marginals efficiently grants us the ability to compute exact conditional distributions efficiently as well. For a PC defined over a set of variables $\mathbf{X}$ and for *any* two disjoint sets of variables $\mathbf{Y} \subset \mathbf{X}$ and $\mathbf{Z} \subset \mathbf{X}$, we have by Bayes rule that $p(\mathbf{Y} \mid \mathbf{Z}) = {}^{p(\mathbf{Y},\mathbf{Z})}/_{p(\mathbf{Z})}$, which can be computed with two passes thorough the network. Thus, PCs can also compute conditionals exactly at a cost that grows only linearly with the model size.

Another relevant property is *determinism* that will be useful in Chapter 7. A PC is called *deterministic* (Darwiche 2003; Vergari et al. 2020) if it holds that for each complete sample $\mathbf{x}$, each sum node has at most one non-zero child. Determinism and decomposability are sufficient conditions for *efficient maximisation* (Peharz et al. 2016), which again, like density evaluation and marginalisation, reduces to a single feedforward pass through the PC graph.

5.1.2 Types of Nodes

In the following we introduce the three types of nodes that constitute a PC.

Distribution nodes

Distribution nodes are the leaves of the graph $\mathcal{G}$. Each distribution node (leaf) ℓ computes a probability density—by an adequate choice of the underlying measure, this also subsumes probability mass functions— over some subset $\mathbf{X}' \subseteq \mathbf{X}$, i.e. a normalised function $p_\ell(\mathbf{x}')\colon \boldsymbol{\mathcal{X}}' \mapsto \mathbb{R}^+$ from the state space of $\mathbf{X}'$ to the non-negative real numbers. Distribution nodes can be arbitrarily complex; for instance, in (Tan and Peharz 2019) the authors considered variational autoencoders (Kingma and Welling 2014) as distribution nodes. However, for a probabilistic query to be tractable in a PC, naturally it must also be tractable in its distribution nodes. That is why distribution nodes, more often than not, compute simple univariate distributions, typically in the exponential family. In fact, some definitions of Sum-Product Networks assume univariate distributions in the leaves, see e.g. (Gens and Domingos 2013).

Sum nodes

Sum nodes compute convex combinations over their children, i.e. if v is a sum node, then v computes

$$p_v(\mathbf{x}) = \sum_{u \in \mathrm{ch}(v)} w_{v,u} p_u(\mathbf{x}), \text{ with } w_{v,u} \geq 0 \text{ and } \sum_{u \in \mathrm{ch}(v)} w_{v,u} = 1.$$

Note that we assume normalised weights in the sum nodes. This, however, does not reduce the expressive power of the model, since for every unnormalised PC that is both smooth and decomposable, we can find a locally normalised PC encoding the same distribution, as shown in Theorem 2 in (Peharz et al. 2015).

We can also interpret sum nodes as encoding discrete latent variables (Peharz et al. 2016). For a sum node v, we can see $p_v(\mathbf{x})$ as computing the marginal of a distribution $p(\mathbf{X}, Z)$, where Z is a discrete random variable assuming values in $\{1, 2, ..., |\operatorname{ch}(v)|\}$ and $w_{v,u_k} = P(Z = k)$ is the weight associated with u_k, the kth child of node v. That is,

$$p_v(\mathbf{x}) = \sum_{k=1}^{|\operatorname{ch}(v)|} P(Z = k) p_{u_k}(\mathbf{x} \,|\, Z = k). \tag{5.2}$$

This interpretation is conceptually useful and generalises the idea of latent variables from (shallow) mixture models to (deep) PCs. It is also of practical relevance since it motivates the Expectation Maximisation (EM) algorithm for PCs (Peharz et al. 2016) as well as representation learning techniques (Vergari et al. 2018). As we shall see in Chapter 8, the latent variable interpretation is also a motivation for our own work (Correia et al. 2022), where we introduce continuous latent variables into PCs.

Product nodes

Product nodes compute the product over their children, i.e. if v is a product node, then

$$p_v(\mathbf{x}) = \prod_{u \in \operatorname{ch}(v)} p_u(\mathbf{x}).$$

Product nodes allow us to represent the idea of statistical independence: the scopes of the children of a product node are pairwise independent. Note that a product node represents a *local* independence relationship that is confined to the sub-tree rooted at the product node. We can also see this is, in fact, a *context-specific* (or *conditional*) independence relationship, where we condition on the latent variables of the ancestor sum nodes.

5.2 Structure Learning

The structure of PCs is endowed with rich probabilistic semantics. Each node in a PC architecture has a meaningful interpretation and is itself a PC computing a valid probability distribution. This opens the way for principled structure learning algorithms.

Unfortunately, structure learning in PCs is still a challenging problem and the literature has mostly focused on parameter learning with random (Peharz et al. 2020b) or hard-coded over-parametrised architectures in deep learning fashion (Peharz et al. 2020a). Yet, we discuss two examples of popular structure learning algorithms for PCs that are relevant to this thesis, namely LearnSPN (Gens and Domingos 2013) that we use in Chapters 6 and 7, and Cutset Networks (Rahman et al. 2014) that are closely related to Generative Forests (Correia et al. 2020b), a class of PCs that we propose in Chapter 6.

5.2.1 LearnSPN

One of the most popular structure learning algorithms for PCs, that will also be relevant to this thesis is LearnSPN (Gens and Domingos 2013). LearnSPN is an intuitive and yet effective algorithm that exploits the very definitions of sum and product nodes. At a high level, the algorithm alternates between i) computing independence relationships between sets of variables and adding a corresponding product node, and ii) clustering observations with respect to some similarity measure and adding a corresponding sum node. Intuitively, if we visualise the data as a matrix of size $n \times m$ (n observations of m variables), we can understand LearnSPN as a sequence of splits where sum nodes correspond to horizontal splits, and product nodes correspond to vertical splits. Such a visualisation of a step of the LearnSPN algorithm is depicted in Figure 5.3.

LearnSPN is fully described in Algorithm 1. The algorithm is presented in a recursive fashion, with three distinct operations.

1. **Add distribution node**: If the dataset $\mathcal{D}_I^{\mathbf{S}}$ comprises a single variable, i.e. $|\mathbf{S}| = 1$, we fit a (parametric) distribution over the dataset and add the corresponding distribution node to the graph. Note that in general we could choose a different stop criterion, such as a maximum depth for the graph $\mathcal{G}$ or a minimum number of variables or instances beyond which we stop splitting. In that case, the distribution nodes might be more complex than univariate distributions.

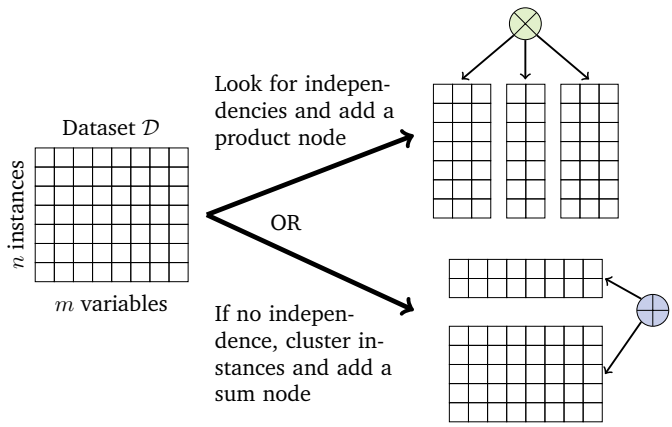

Figure 5.3: Illustration of a step of the LearnSPN algorithm. Given a dataset $\mathcal{D}$, either split it 'vertically' with a product node or 'horizontally' with a sum node.

2. **Add product node**: Otherwise, we search for a partition of variables in $\mathbf{S}$, such that each subset $\mathbf{S}_i \subset \mathbf{S}$ is statistically independent of the remaining variables $\mathbf{S} \setminus \mathbf{S}_i$ given data $\mathcal{D}_I^{\mathbf{S}}$, with

$$\bigcup \mathbf{S}_i = \mathbf{S}, \quad \mathbf{S}_i \cap \mathbf{S}_j = \emptyset, \quad \forall i \neq j.$$

   This is often done via pairwise independence tests, requiring the user to define appropriate test[1] and p-value. Once we have found the subsets of independent variables in $\mathcal{D}_I^{\mathbf{S}}$, we add a product with one child for each of the these subsets. Note that the number of independent subsets (equiv. children of the product node) is not fixed a priori but allowed to vary from 1 (no independence relation found, go to next step) to $|\mathbf{S}|$.

3. **Add sum node**: If we find no independence relations, we add a sum node by clustering the data instances in $\mathcal{D}_I^{\mathbf{S}}$ into K subsets $\{\mathcal{D}_{I_k}^{\mathbf{S}}\}_{k=1}^K$ with

$$\bigcup_{k=1}^{K} I_k = I, \quad I_i \cap I_j = \emptyset \quad \forall i \neq j.$$

[1] Common choices are Kruskal-Wallis (Kruskal and Wallis 1952), Kendall's Tau (Kendall 1938), Chi-Square (Pearson 1900) or G-test (McDonald 2009).

In the original implementation(Gens and Domingos 2013), this is done via hard Expectation Maximisation, but any clustering algorithm that is pertinent to the data at hand would be applicable. In general, this step also requires a hyperparameter in the form of the number of clusters K. Once we have clustered the instances, we add a sum node to the graph $\mathcal{G}$, where each child corresponds to one cluster. It is worth noticing that the maximum likelihood parameters of such a sum node are given by the relative size of each cluster, i.e. $w_k = |I_k|/|I|$, and thus are trivial and readily obtained when running LearnSPN.

Albeit effective, LearnSPN has a few drawbacks. The most relevant in practice is its high computational cost due to the expensive pairwise independence tests, which severely hinders its application to datasets with many variables. LearnSPN is also based on heuristics and does not optimise the *global* likelihood directly. Thus, the only guarantee we can derive from it is that the learnt PC will be *locally* optimal in the maximum likelihood sense (Gens and Domingos 2013), i.e. *local* perturbations of its parameters do not yield a PC with a higher likelihood.

Input : Training data $\mathcal{D}_I^{\mathbf{S}}$ with $\mathbf{S} \subseteq \mathbf{X}$ and number of clusters K
Output: Probabilistic Circuit $\mathcal{G}$
Function `LearnSPN(`$\mathcal{D}$`)`**:**
  if $|S|$ = *1* **then**
    `// Dataset contains a single variable`
    return a distribution node learned over $\mathcal{D}_I^{\mathbf{S}}$
  else
    try
      `// Try to add a product node`
      partition $\mathcal{D}$ into approximately independent subsets S_i
      `// with` $\bigcup \mathbf{S}_i = \mathbf{S}$ `and` $\mathbf{S}_i \cap \mathbf{S}_j = \emptyset$ `for` $i \neq j$
      return $\prod_i$`LearnSPN` $(\mathcal{D}_I^{\mathbf{S}_i})$
    catch
      `// No independent sets of variables, add a sum node`
      partition $\mathcal{D}$ into K subsets of similar instances $\{I_k\}_{k=1}^K$
      `// with` $\bigcup_{k=1}^K I_k = I$ `and` $I_i \cap I_j = \emptyset$ `for` $i \neq j$
      return $\sum_{k=1}^K \frac{n_k}{n}$`LearnSPN` $(\mathcal{D}_{I_k}^{\mathbf{S}})$
  end

Algorithm 1: LearnSPN (Gens and Domingos 2013).

5.2.2 Cutset Networks

Cutset Networks (or CNets) (Rahman et al. 2014) is another popular type of Probabilistic Circuits, where structure and parameters are learned together. Its corresponding learning algorithm, LearnCNet, does not require clustering algorithms or expensive independence tests like LearnSPN. Instead, LearnCNet relies on *splitting heuristics* that partition the data to greedily optimise some metric like the joint entropy. In that regard, CNets are similar in spirit to our work on Generative Forests that we introduce in the Chapter 6.

CNets are based on OR-trees (Pearl 1988; Dechter and Mateescu 2007), tree structures representing the search space in probabilistic inference. To be precise, each node in an OR-tree is associated with a variable X_i and each of its edges represents conditioning on one of the possible instantiations $X_i = k$ with $k \in \mathcal{X}_i$. Some authors have studied SPNs without latent variables which are quite similar to OR-trees (Dennis and Ventura 2015). In this kind of SPN, each child of a sum node represents conditioning on an observed rather than latent variable, as discussed in Section 5.1.2. In contrast to latent sum nodes (Equation 5.2), a node associated to variable X_i in an OR-tree computes

$$p_v(\mathbf{x}) = \sum_{k \in \mathcal{X}_i} P(X_i = k) p_{u_k}(\mathbf{x} \mid X_i = k).$$

The idea of CNets is to learn a partition of the input space and subsequently fit a Chow-Liu tree[2] to each of the cells defined by such a partition, as shown in Algorithm 2. For ease of presentation, we outlined the algorithm considering binary variables only, but LearnCNet is readily applicable to arbitrary categorical data. The algorithm works recursively, partitioning the data until some stop criterion is met, like a minimum number of instances per cell so that the we have enough data in each cell to fit a Chow-Liu tree.

In Algorithm 2 we do not specify the criterion used to choose a variable to split on, since different heuristics might apply. In (Rahman et al. 2014) the authors propose to greedily minimise the joint entropy under the empirical distribution $\hat{P}_{\mathcal{D}}$ induced by some data $\mathcal{D}$

$$\mathrm{H}(\mathcal{D}_I^{\mathbf{S}}) = \frac{1}{|\mathbf{S}|} \sum_{X_i \in \mathbf{S}} \mathrm{H}(\mathcal{D}_I^{X_i}),$$

[2]See Chapter 2 for a brief discussion on Chow-Liu trees.

where $\mathrm{H}_{\mathcal{D}}(X_i)$ is the entropy of a single variable

$$\mathrm{H}(\mathcal{D}_I^{X_i}) = -\sum_{k \in \mathcal{X}_i} \hat{P}_{\mathcal{D}}(X_i = k) \log \hat{P}_{\mathcal{D}}(X_i = k)$$

$$\hat{P}_{\mathcal{D}}(X_i = k) = \frac{\left|\mathcal{D}_{[X_i=k]}\right|}{|\mathcal{D}|} = \frac{|\{\mathbf{x} \in \mathcal{D} : \mathbf{x}[i] = k\}|}{|\mathcal{D}|}.$$

At each step, we compute the relative gain (reduction of entropy) we would get by splitting in each one of the variables

$$\mathrm{gain}_{\mathcal{D}_I^{\mathbf{S}}}(X_i) = \mathrm{H}(\mathcal{D}_I^{\mathbf{S}}) - \sum_{k \in \mathcal{X}_i} \hat{P}_{\mathcal{D}}(X_i = k) \mathrm{H}\left(\mathcal{D}_{[X_i=k]}^{\mathbf{S} \setminus X_i}\right).$$

We then create a node associated with the variable yielding the largest relative gain and add one edge for each of possible instantiations of that variable.

LearnCNet is an effective structure learner, but unfortunately it is not trivially extendable to continuous data and can still be expensive in scenarios with

Input : Training data $\mathcal{D}_I^{\mathbf{S}}$ with $\mathbf{S} \subseteq \mathbf{X}$
Output: Probabilistic Circuit $\mathcal{G}$
Function `LearnCNet(`$\mathcal{D}$`)`**:**
  if *Termination condition reached* **then**
    `// Return a CLT learnt over` $\mathcal{D}_I^{\mathbf{S}}$
    return ChowLiuTree($\mathcal{D}_I^{\mathbf{S}}$)
  end
  Create a new sum node v
  Heuristically choose a variable X_i in $\mathbf{S}$ to split on
  `// Split the index set` I `into two according to` X_i
  Let $I_{left} = \{i : \mathbf{x}[i] = 0, \mathbf{x} \in \mathcal{D}_I^{\mathbf{S}}\}$
  Let $I_{right} = \{i : \mathbf{x}[i] = 1, \mathbf{x} \in \mathcal{D}_I^{\mathbf{S}}\}$
  `// Create` $\mathrm{ch}(v)$ `by recursively calling LearnCNet`
  $u_{left} \leftarrow$ `LearnCNet` $(\mathcal{D}_{I_{left}}^{\mathbf{S} \setminus X_i})$
  $u_{right} \leftarrow$ `LearnCNet` $(\mathcal{D}_{I_{right}}^{\mathbf{S} \setminus X_i})$
  `// Set the weights of` v
  $w_{v,u_{left}} \leftarrow |I_{left}|/|I|$
  $w_{v,u_{right}} \leftarrow |I_{right}|/|I|$
  return v

Algorithm 2: LearnCNet (Rahman et al. 2014).

many variables or large state spaces. Our work on Generative Forests (GeFs) addresses these issues, since GeFs are applicable to datasets of mixed data types (continuous and discrete) and use more efficient splitting heuristics. The structure learning algorithm in GeFs, however, minimise the classification error with respect to a specific variable rather than optimise the joint log-likelihood like CNets.

5.3 Parameter Learning

A number of structure learning algorithms for PC, like LearnSPN (Gens and Domingos 2013) and LearnCNet (Rahman et al. 2014) discussed in the previous section, also learn the parameters. However, it is also possible to learn or posit a structure and then optimise the parameters to fit some data. In particular, Probabilistic Circuits are fully differentiable computational graphs like neural networks, and thus we are free to use any gradient descent optimisation algorithm to learn the parameters of a PC. Algorithms such as Adam (Kingma and Ba 2014), Adagrad (Duchi et al. 2011) and Adadelta (Zeiler 2012) are all popular stochastic gradient descent (SGD) variants designed for neural networks that are readily applicable to PCs. Nonetheless, the most common and often most effective way to train PCs is via Expectation Maximisation (EM) (Poon and Domingos 2011; Peharz et al. 2016, 2020a).

The EM algorithm for PCs (Peharz et al. 2016) hinges on the latent variable interpretation discussed when we introduced sum nodes in Section 5.1. Exploiting the differential approach to inference (Darwiche 2003), it can be shown that the EM update for the weight of a sum node $w_{v,u}$ can be written down as follows (Peharz et al. 2016)

$$
\begin{aligned}
w_{v,u} &\leftarrow \frac{\sum_{\mathcal{D}} w_{v,u}\, n_{v,u}(\mathbf{x})}{\sum_{\mathcal{D}, u \in \mathrm{ch}(v)} n_{v,u}(\mathbf{x})} \\
n_{v,u}(\mathbf{x}) &= \frac{1}{p_r(\mathbf{x})} \frac{\partial p_r(\mathbf{x})}{\partial p_v(\mathbf{x})} p_u(\mathbf{x}),
\end{aligned}
\tag{5.3}
$$

where $n_{v,u}(\mathbf{x})$ is an expected statistic and the sum ranges over all instances $\mathbf{x}$ in data $\mathcal{D}$. A similar update can be derived for distribution (leaf) nodes. If we consider distribution nodes defining a distribution in the exponential family, $p(\mathbf{x} \mid \boldsymbol{\theta}) = h(\mathbf{x}) \exp\{\eta(\boldsymbol{\theta}) \cdot T(\mathbf{x}) - A(\boldsymbol{\theta})\}$, where $\eta(\boldsymbol{\theta})$ are the natural parameters, h is a base measure, T is the sufficient statistic and A is the log-partition

function, then we can compute the updates on $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}_\ell \leftarrow \frac{\sum_{\mathcal{D}} p_\ell(\mathbf{x}) T(\mathbf{x})}{\sum_{\mathcal{D}} p_\ell(\mathbf{x})}$$
$$n_\ell(\mathbf{x}) = \frac{1}{p_r(\mathbf{x})} \frac{\partial p_r(\mathbf{x})}{\partial p_\ell(\mathbf{x})} p_\ell(\mathbf{x}), \tag{5.4}$$

where $n_\ell(\mathbf{x})$ are again expected statistics in the EM algorithm.

The EM algorithm in its original form computes expected statistics over the entire dataset, which is typically computationally expensive and often impracticable (Bottou 1998). That is why one often resorts to stochastic EM algorithms (Sato 1999), which are akin SGD. Essentially, we compute expected statistics over a mini-batch of the data and update the parameters

$$w_{v,u} \leftarrow (1-\lambda) w_{v,u} + \lambda w_{v,u}^{\text{mb}} \qquad \boldsymbol{\theta}_\ell \leftarrow (1-\lambda)\, \boldsymbol{\theta}_\ell + \lambda\, \boldsymbol{\theta}_\ell^{\text{mb}},$$

where λ is a step-size hyperparameter, and $w_{v,u}^{\text{mb}}$ and $\boldsymbol{\theta}_\ell^{\text{mb}}$ are computed as in Equations 5.3 and 5.4, respectively, but over a mini batch rather than the entire dataset. Furthermore, for distributions in the exponential family, the stochastic EM can be seen as performing SGD on the natural parameters (Sato 1999), which is known to speed up convergence (Amari 1998).

5.4 Probabilistic Circuits and Bayesian Networks

Finally, we conclude this chapter by exploring the relationship between Bayesian Networks and Probabilistic Circuits, the two classes of probabilistic generative models that we study in this thesis.

In terms of their expressive power, Bayesian Networks and Probabilistic Circuits over *discrete data* are equivalent (Poon and Domingos 2011): both can represent any joint distribution over discrete random variables, and there are algorithms to convert between the two models. We can map a discrete PC into a BN in time and space linear in the size of the PC; and we can map a discrete BN to a PC in time and space that might be exponential in the tree-width of the BN (Zhao et al. 2015). In fact, the original motivation for the introduction of network polynomials, which are closely related to Probabilistic Circuits, was computing inference routines in Bayesian Networks (Darwiche 2003). That said, the conversion between BNs and PCs has been largely confined to theoretical developments, with only a few notable practical applications, like Hidden Chow-Liu Tree PC architectures (Liu and Van den Broeck 2021).

In the case of continuous variables, Bayesian Networks are actually more expressive. To see why, consider a simple BN with two continuous random variables X_1 and X_2 such that X_1 is the parent of X_2, i.e. $X_1 \rightarrow X_2$. It is easy to see that, except for corner cases such as linear-Gaussian models (Roweis and Ghahramani 1999), the distribution $p(X_1, X_2) = p(X_2|X_1)p(X_1)$ cannot be captured by a PC because we would need a sum node with an infinite number of children, one for each possible instantiation of X_1, so that we can represent $p(X_2|X_1)$ exactly. That limitation is, in fact, one of the motivations for introducing continuous latent variables to PCs, as discussed in Chapter 8.

The key distinction between BNs and PCs is that the former focus on representing statistical relations among variables, while the latter is designed to evaluate distributions efficiently. One of the challenges in running inference on BNs is that their network polynomials have size exponential in the number of variables. The advantage of PCs is that they encode and evaluate a network polynomial more efficiently, in polynomial space and time (Poon and Domingos 2011). Unfortunately, it is known that not every joint distribution can be represented by a PC of size polynomial in the number of variables (Martens and Medabalimi 2014), and thus PC research rests on the assumption that polynomial-size PCs are already enough to capture many or most of the distributions of interest.

5.5 Conclusion

In this chapter we have given a general introduction to Probabilistic Circuits covering the basic definitions of the framework as well as learning algorithms that are relevant to our work. This brief introduction should be enough to allow the reader to follow the main contributions in this thesis with ease, but for the interested reader we recommend (Choi et al. 2020) and (Vergari et al. 2020) for general introductions that are more comprehensive than this one.

Chapter 6

Generative Forests

Decision Trees (DTs) and Random Forests (RFs) are powerful discriminative learners and tools of central importance to the everyday machine learning practitioner and data scientist. Due to their discriminative nature, however, they lack principled methods to process inputs with missing features or to detect outliers, and thus are often paired with imputation techniques or a separate generative model. In this work, we demonstrate that DTs and RFs can naturally be interpreted as generative models, by drawing a connection to Probabilistic Circuits, a prominent class of tractable probabilistic models. This reinterpretation equips them with a full joint distribution over the feature space and leads to Generative Decision Trees (GeDTs) and Generative Forests (GeFs), a family of novel hybrid generative-discriminative models. This family of models retains the overall characteristics of DTs and RFs while additionally being able to handle missing features by means of marginalisation. Under certain assumptions, frequently made for Bayes consistency results, we show that consistency in GeDTs and GeFs extend to any pattern of missing input features, if missing at random. Empirically, we show that our models often outperform common routines to treat missing data, such as K-nearest neighbour imputation, and moreover, that our models can naturally detect outliers by monitoring the marginal probability of input features.

This chapter is almost integrally based on Alvaro Correia, Robert Peharz and Cassio de Campos: Joints in Random Forest, *Advances in Neural Information Processing Systems (NeurIPs)*, 2020.

6.1 Introduction

Decision Trees (DTs) and Random Forests (RFs) are probably the most widely used non-linear machine learning models of today. While Deep Neural Networks are in the lead for image, video, audio, and text data—likely due to their beneficial inductive bias for signal-like data—DTs and RFs are, by and large, the default predictive model for tabular, domain-agnostic datasets. Indeed, Kaggle's 2019 report on the *State of Data Science and Machine Learning* (Kaggle 2019) lists DTs and RFs as second most widely used techniques, right after linear and logistic regressions. Moreover, a study by Fernandez et al. (Fernández-Delgado et al. 2014) found that RFs performed best on 121 UCI datasets against 179 other classifiers. Thus, it is clear that DTs and RFs are of central importance for the current machine learning practitioner.

DTs and RFs are generally understood as *discriminative models*, that is, they are solely interpreted as predictive models, such as *classifiers* or *regression functions*, while attempts to additionally interpret them as *generative* models are scarce. In a nutshell, the difference between discriminative and generative models is that the former aim to capture the *conditional distribution* $P(Y \mid \mathbf{X})$, while the latter aim to capture the whole *joint distribution* $P(Y, \mathbf{X})$, where $\mathbf{X}$ are the input features and Y is the set of variables to be predicted—discrete for classification and continuous for regression. In this chapter, we focus on classification tasks, but it is also possible to extend our methods to regression tasks[1].

Generative and discriminative models are rather complementary in their strengths and use cases. While discriminative models typically fare better in predictive performance, generative models allow us to analyse and capture the structure present in the input space. They are also 'all-round predictors', that is, not restricted to a single prediction task but also capable of predicting any X given $Y \cup \mathbf{X} \setminus X$. Moreover, generative models have some crucial advantages on the prediction task $P(Y \mid \mathbf{X})$ a discriminative model has been trained on, as they naturally allow us to *detect outliers* (by monitoring $P(\mathbf{X})$) and *treat missing features* (by marginalisation). A purely discriminative model does not have any 'innate' mechanisms to deal with these problems, and needs to be supported by a generative model $P(\mathbf{X})$ (to detect outliers) or imputation techniques (to handle missing features).

Ideally, we would like the best of both worlds: having the good predictive performance of discriminative models *and* the advantages of generative models.

[1] The main challenge in the regression case is handling missing data since propagating a continuous distribution through the graph requires more sophisticated data structures (and assumptions) than propagating categorical distributions. See Section 6.7 for a discussion on this topic.

In this chapter, we show that this is achievable for DTs and RFs by relating them to Probabilistic Circuits. There have been other methods in the PC literature that are closely related to DTs, with Cutset Networks (or CNets) being a notable example that can be seen as a type of generative DT (Rahman et al. 2014). However, the connection to classical, discriminative DTs (Quinlan 1986) and RFs (Breiman 2001) had been left unexplored until the introduction of Generative Trees and Forests, the models that we discuss in this chapter.

We show that DTs and RFs can be naturally cast into the PC framework. For any given DT, we can construct a corresponding PC, a *Generative Decision Tree* (GeDT), representing a full joint distribution $P(Y, \mathbf{X})$. This distribution gives rise to the predictor

$$P(Y \mid \mathbf{X}) = \frac{P(Y, \mathbf{X})}{\sum_y P(y, \mathbf{X})},$$

which is identical to the original DT, if we impose certain constraints on the conversion from DT to GeDT.

Additionally, a GeDT also fits the joint distribution $P(\mathbf{X})$ to the training data, 'upgrading' the DT to a fully generative model. For a *completely observed sample* $\mathbf{X} = \mathbf{x}$, the original DT and a corresponding GeDT agree entirely (yield the exact same predictions), and moreover have the *same computational complexity*, since standard DT evaluation can be shown to be a 'shortcut' routine to compute $P(Y \mid \mathbf{X})$ in the PC. See Section 6.6 for a discussion on the computational complexity of GeDTs. By converting each DT in an RF into an GeDT, we obtain an ensemble of GeDTs, which we call *Generative Forest* (GeF). Clearly, if each GeDT in a GeF agrees with its original DT, then GeFs also agree with their corresponding RFs.

GeDTs and GeFs have a crucial advantage in the case of *missing features*, that is, assignments $\mathbf{X}_o = \mathbf{x}_o$ for some subset $\mathbf{X}_o \subset \mathbf{X}$, while $\mathbf{X}_{\neg o} = \mathbf{X} \setminus \mathbf{X}_o$ are *missing at random*. In a GeDT, we can marginalise the missing features and yield the predictor

$$P(Y \mid \mathbf{X}_o) = \frac{\int_{\mathbf{x}_{\neg o}} P(Y, \mathbf{X}_o, \mathbf{x}_{\neg o}) \mathrm{d}\mathbf{x}_{\neg o}}{\sum_y \int_{\mathbf{x}_{\neg o}} P(y, \mathbf{X}_o, \mathbf{x}_{\neg o}) \mathrm{d}\mathbf{x}_{\neg o}}. \tag{6.1}$$

For GeFs, we get a corresponding ensemble predictor for missing features, by applying marginalisation to each GeDT. Using the true data generating distribution in Equation 6.1 would deliver the *Bayes optimal predictor* for any subset $\mathbf{X}_o$ of observed features. Thus, since GeDTs are trained to approximate the true distribution, using the predictor of Equation 6.1 under missing data is well

justified. We show GeDTs are in fact *consistent*: they converge to the Bayes optimal classifier as the number of data points goes to infinity. Our proof requires similar assumptions to those of previous results for DTs (Biau et al. 2008; Breiman et al. 1984; Gordon and Olshen 1978) but is substantially more general: while consistency in DTs is shown only for a classifier $P(Y \mid \mathbf{X})$ using fully observed samples, our consistency result holds for *all* $2^{|X|}$ classifiers $P(Y \mid \mathbf{X}_o)$: one for each observation pattern $\mathbf{X}_o \subseteq \mathbf{X}$. While the high-dimensional integrals in Equation 6.1 seem prohibitive, they are in fact *tractable*, since a remarkable feature of PCs is that computing any marginal has *the same complexity* as evaluating the full joint, namely linear in the *circuit size*.

This ability of our models is desirable, as there is no clear consensus on how to deal with missing features in DTs at test time[2]: the most common strategy is to use *imputation*, e.g. *mean* or *k-nearest-neighbour (KNN) imputation*, and subsequently feed the completed sample to the classifier. DTs also have two 'built-in' methods to deal with missing features that do not require external models. These are the so-called *surrogate splits* (Therneau et al. 1997) and an unnamed method proposed by Friedman in 1977 (Friedman 1977; Quinlan 1987a). See Appendix A for a detailed description of each of these methods to handle missing data. Among these, KNN imputation seems to be the most widely used, and typically delivers good results on real-world data. However, we demonstrate it does not lead to a consistent predictor under missing data, even when assuming idealised settings. Moreover, in our experiments, we show that GeF classification under missing inputs often outperforms standard RFs with KNN imputation.

Our generative interpretation can be easily incorporated in existing DT learners and does not require drastic changes in the learning and application practice of DTs and RFs. Essentially, any DT algorithm can be used to learn GeDTs, requiring only minor bookkeeping and some extra generative learning steps. There are de facto no model restrictions concerning the additional generative learning steps, representing a generic scheme to augment DTs and RFs to generative models.

6.2 Notation and Background

In this section, we present the notation and necessary background on GeFs. The notation is congruent with the rest of thesis, but we reintroduce it in the context

[2] In this chapter, we focus on missing features during test time (training set is assumed complete).

of classification tasks for the sake of completeness. To this end, let the set of explanatory variables (features) be $\mathbf{X} = \{X_1, X_2, \ldots, X_m\}$, where continuous X_i assume values in some compact set $\mathcal{X}_i \subset \mathbb{R}$ and discrete X_i assume values in $\mathcal{X}_i = \{1, \ldots, K_i\}$, where K_i is the number of states for X_i. Let the joint *feature space* of $\mathbf{X}$ be denoted as $\boldsymbol{\mathcal{X}}$. We denote joint states, i.e. elements from $\boldsymbol{\mathcal{X}}$, as $\mathbf{x}$ and let $\mathbf{x}[i]$ be the state in $\mathbf{x}$ belonging to X_i. The class variable is denoted as Y, assuming values in $\mathcal{Y} = \{1, \ldots, K\}$, where K is the number of classes. We assume that the pair $(\mathbf{X}, Y)$ is drawn from a fixed joint distribution $\mathbb{P}^*(\mathbf{X}, Y)$ which has density $p^*(\mathbf{X}, Y)$. While the true distribution $\mathbb{P}^*$ is unknown, we assume that we have a dataset $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ of n i.i.d. samples from $\mathbb{P}^*$. When describing a directed graph $\mathcal{G}$, we refer to its set of nodes as V, reserving letters u and v for individual nodes. We denote the set of children and parents of a node v as $\mathrm{ch}(v)$ and $\mathrm{pa}(v)$, respectively. Nodes ℓ without children are referred to as *leaves*, and nodes r without parents are referred to as *roots*.

Decision Trees

A *decision tree* (DT) is based on a *rooted directed tree* $\mathcal{G}$, i.e. an acyclic directed graph with exactly one root v_r and whose other nodes have exactly one parent. Each node v in the DT is associated with a *cell* $\boldsymbol{\mathcal{X}}_v$, which is a subset of the feature space $\boldsymbol{\mathcal{X}}$. The cell of the root node r is the whole $\boldsymbol{\mathcal{X}}$. The *child cells* of node v form a partition of $\boldsymbol{\mathcal{X}}_v$, that is,

$$\bigcup_{u \in \mathrm{ch}(v)} \boldsymbol{\mathcal{X}}_u = \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{X}}_u \cap \boldsymbol{\mathcal{X}}_{u'} = \emptyset, \ \forall u, u' \in \mathrm{ch}(v).$$

These partitions are usually defined via *axis-aligned splits*, by associating a decision variable X_i to v, and partitioning the cell according to some rule on X_i's values. Formally, we first project $\boldsymbol{\mathcal{X}}_v$ onto its ith coordinate, yielding $\mathcal{X}_{i,v} := \{\mathbf{x}[i] \mid \mathbf{x} \in \boldsymbol{\mathcal{X}}_v\}$, and construct a partition $\{\mathcal{X}_{i,u}\}_{u \in \mathrm{ch}(v)}$ of $\mathcal{X}_{i,v}$. The child cells are then given by $\boldsymbol{\mathcal{X}}_u = \{\mathbf{x} \mid \mathbf{x} \in \boldsymbol{\mathcal{X}}_v \wedge \mathbf{x}[i] \in \mathcal{X}_{i,u}\}$. Thus, each node in a DT represents a *decision* based on its associated variable X_i. Common choices for this partition are *full splits* for discrete variables, and *thresholding* for continuous variables, with

$$\{\mathcal{X}_{i,u}\}_{u \in \mathrm{ch}(v)} = \begin{cases} \{\{x_i\}\}_{x_i \in \mathcal{X}_{i,v}} & \text{full split} \\ \{\{x_i < t\}, \{x_i \geq t\}\} & \text{thresholding,} \end{cases}$$

where children u and states x_i are in one-to-one correspondence, and t some learnable threshold in $\mathbb{R}$. In this work, we also use binary splits for discrete

variables instead of full splits; we learn $\mathcal{X}_{i,u}$ directly by considering all possible 2^{K_i} splits. Such a split is more expensive to compute than a full split, but that is compensated by more compact and potentially more effective tree structures. The clustering induced by these learnable binary splits might be meaningful and avoid unnecessary splits that could lead to cells with very few datapoints, hurting density estimation later on (see Section 6.4). At any rate, if the full split is optimal, we can recover it by a sequence of binary splits. Note that the leaf cells of a DT form a partition $\boldsymbol{\mathcal{A}}$ of the feature space $\boldsymbol{\mathcal{X}}$. We denote the elements of $\boldsymbol{\mathcal{A}}$ as $\mathcal{A}$ and define $\mathcal{A}_\ell = \boldsymbol{\mathcal{X}}_\ell$ for each leaf ℓ. If all features are continuous, the leaves of a DT represent a partition of $\boldsymbol{\mathcal{X}}$ into a set of rectangular cells $\boldsymbol{\mathcal{A}}$.

A *DT classifier* is constructed by equipping each $\mathcal{A} \in \boldsymbol{\mathcal{A}}$ with a classifier $f^{\mathcal{A}} \colon \mathcal{A} \mapsto \Delta^K$, where Δ^K is the set of probability distributions over K classes, i.e. $f^{\mathcal{A}}$ is a conditional distribution defined on $\mathcal{A}$. This distribution is typically stored as absolute class counts of the training samples contained in $\mathcal{A}$.

The overall DT classifier is given as $f(\mathbf{x}) = f^{\mathcal{A}(\mathbf{x})}(\mathbf{x})$ where $\mathcal{A}(\mathbf{x})$ is the leaf cell containing $\mathbf{x}$; $\mathcal{A}(\mathbf{x})$ is found by parsing the DT top-down, following the partitions (decisions) consistent with $\mathbf{x}$. This formulation captures the vast majority of DT classifiers proposed in the literature, notably CART (Breiman et al. 1984) and ID3 (Quinlan 1986). The probably most widely used variant of DTs—which we also assume in this chapter—is to define $f^{\mathcal{A}}$ as a constant function, returning the class proportions in cell $\mathcal{A}$, which translates into a categorical distribution over classes. The $\arg\max$ of $f^{\mathcal{A}}(\mathbf{x})$ is equivalent to majority voting among all training samples which fall into the same cell. When learning a DT, the number of available training samples per cell reduces quickly, which leads to overfitting and justifies the need for pruning techniques (Breiman et al. 1984; Mingers 1987; Quinlan 1986, 1987b).

Random Forests

Random Forests (RFs) are ensembles of DTs which effectively counteract overfitting. Each DT in a RF is learnt in a randomised fashion by, at each learning step, drawing a random sub-selection of variables containing only a fraction p of all variables, where typical values are $p = 0.3$ or $p = \sqrt{m}$. The resulting DTs are not pruned but made 'deep' until each leaf cell contains either only samples of one class or less than T samples, where typical values are $T \in \{1, 5, 10\}$. This yields low bias, but high variance in the randomised DTs, which makes them good candidates for *bagging* (bootstrap aggregation) (Hastie et al. 2009). Thus, to further increase the variability among the trees, each of them is learnt on a *bootstrapped* version of the training data (Breiman 2001).

6.3 Related Work

Among the many variations of DTs and RFs that have been proposed in the last decades, the closest to our work are those that, similarly to GeDTs and GeFs, extend DT leaves with 'non-trivial' models. Notable examples are DTs where the leaves are modelled by linear and logistic regressors (Frank et al. 1998; Landwehr et al. 2005; Quinlan et al. 1992), kernel density estimators (KDEs) (Loh 2009; Smyth et al. 1995), linear discriminant models (Gama et al. 2004; Kim and Loh 2003), KNN classifiers (Buttrey and Karo 2002; Loh 2009), and Naive-Bayes classifiers (NBCs) (Kohavi 1996). Nonetheless, all these previous works focus primarily on improving the classification accuracy or smoothing probability estimates but do not model the full joint distribution, like in this work. Even extensions by Smyth et al. (Smyth et al. 1995) and Kohavi (Kohavi 1996), which include generative models (KDEs and NBCs, respectively) do not exploit their generative properties. To the best of our knowledge, GeFs are the first DT framework that effectively model and leverage the full joint distribution in a classification context. That is of practical significance as none of these earlier extensions of DTs offer a principled way to treat missing values or detect outliers. Here it is also worth mentioning the contemporary work by Khosravi et al.; they propose similar approaches to extend models to a full joint (Khosravi et al. 2019) and handle missing data in DTs (Khosravi et al. 2020).

On the other side of the spectrum, DTs have also been extended to density estimators (Gray and Moore 2003; Rahman et al. 2014; Ram and Gray 2011; Wu et al. 2014). Among these, Density Estimation Trees (DETs) (Ram and Gray 2011), Cutset Networks (CNets) (Rahman et al. 2014), and randomised ensembles thereof (Di Mauro et al. 2017), are probably the closest to our work. These models are trained with a greedy tree-learning algorithm but minimise a modified loss function that matches their generative nature: joint entropy across all variables in CNets, mean integrated squared error in DETs. Notably, CNets, like GeFs, are Probabilistic Circuits, and hence also allow for tractable inference and marginalisation. They, however, have not been applied in a discriminative setting and are not backwards compatible with DTs and RFs. Moreover, GeDTs (and GeFs) can be seen as a family of models depending on the estimation at the leaves, making a clear parallel with what DTs (and RFs) offer.

Finally, one can also mimic the benefits of generative models in ensembles by learning predictors for all variables, as in MERCS (Wolputte et al. 2018). That is fundamentally different from our probabilistic approach and might entail prohibitively large numbers of predictors. Handling missing values, in the worst case, would require one predictor for each of the $2^{|\mathbf{X}|}$ missing patterns, and that

is why MERCS relies on imputation methods when needed. Conversely, GeDTs model a full joint distribution, thus being more compact and interpretable.

6.4 Generative Decision Trees

Given a learnt DT and the dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ it has been trained on, we can obtain a corresponding generative model, by converting the DT into a PC. This conversion is given in Algorithm 3.

Input : Decision Tree $\mathcal{G}$ and training data $\mathcal{D}$
Output: Probabilistic Circuit $\mathcal{G}'$
let $\mathcal{G}'$ be a structural copy of $\mathcal{G}$, and let $v' \in V'$ be the node in $\mathcal{G}'$ which corresponds to $v \in V$ of $\mathcal{G}$
for the root node r, set $\mathcal{D}_r = \mathcal{D}$
for v **in** *topdownsort(V)* **do**
  if v *is internal* **then**
    get partition $\mathcal{X}_{i,u}$ of decision variable X_i associated with v
    for $u \in \mathrm{ch}(v)$ **do**
      let $w_{v'u'} = \frac{\sum_{\mathbf{x} \in \mathcal{D}_v} \mathbb{1}(\mathbf{x}[i] \in \mathcal{X}_{i,u})}{|\mathcal{D}_v|}$
      set $\mathcal{D}_u = \{\mathbf{x} \in \mathcal{D}_v \mid \mathbf{x}[i] \in \mathcal{X}_{i,u}\}$
    end
    let v' be a sum node $\sum_{u' \in \mathrm{ch}(v')} w_{v'u'} u'$
  else
    let v' be a density $p_{v'}(\mathbf{x}, y)$ with support $\mathcal{A}_v$, learnt from $\mathcal{D}_v$
  end
end

Algorithm 3: Converting DT to PC (GeDT).

In a nutshell, Algorithm 3 converts each decision node into a sum node and each leaf into a density with support restricted to the leaf's cell. The training samples can be figured to be routed from the root node to the leaves, following the decisions at each decision/sum node. The sum weights are given by the fraction of samples which are routed from the sum node to each of its children. The leaf densities are learnt on the data which arrives at the respective leaves.

As an example, assuming $\mathbf{X}$ and Y factorise at the leaves, Algorithm 3 applied to the DT on the left-hand side of Figure 6.1 gives the PC on the right-hand

side and the following densities at the leaves:

$$
\begin{aligned}
p_1(X_1, X_2, Y) &= p_1(X_1, X_2)(0 \cdot \mathbb{1}(Y = 0) + 1 \cdot \mathbb{1}(Y = 1)), \\
p_2(X_1, X_2, Y) &= p_2(X_1, X_2)(0.25 \cdot \mathbb{1}(Y = 0) + 0.75 \cdot \mathbb{1}(Y = 1)), \\
p_3(X_1, X_2, Y) &= p_3(X_1, X_2)(1 \cdot \mathbb{1}(Y = 0) + 0 \cdot \mathbb{1}(Y = 1)),
\end{aligned}
$$

Note that X_1 is deterministic (all mass absorbed in one state) in p_1 and p_2, since X_1 has been fixed by the tree construction, while p_3 is a 'proper' distribution over X_1 and X_2. Densities $p_i(X_1, X_2)$ do not appear in the DT representation and illustrate the extension brought in by the PC formalism.

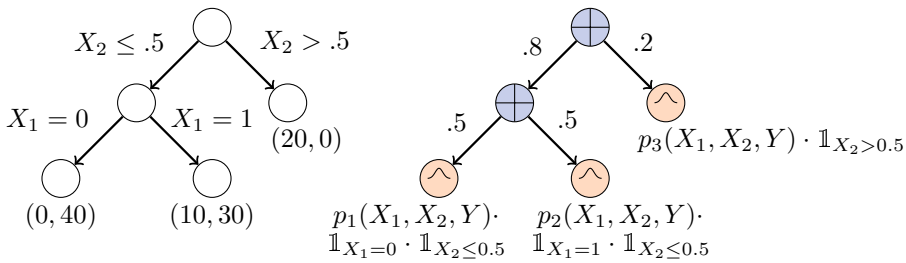

Figure 6.1: Illustration of a DT and its corresponding PC as obtained by Algorithm 3.

We denote the output of Algorithm 3 as a *Generative Decision Tree* (GeDT). Note that GeDTs are proper PCs over $(\mathbf{X}, Y)$, albeit rather simple ones: they are tree-shaped and contain only sum nodes. They are clearly smooth, since each leaf density has the full scope $(\mathbf{X}, Y)$, and they are trivially decomposable, as they do not contain products. Thus, both the full density or any sub-marginal can be evaluated by simply evaluating the GeDT bottom up, where for marginalisation tasks we first need to perform marginalisation at the leaves. Moreover, as shown in (Peharz et al. 2014; Rahman et al. 2014), the sum-weights set by Algorithm 3 are in fact the *maximum likelihood* weights for deterministic PCs. Finally, it is also easy to show that any GeDT is *deterministic*.

Proposition 1
A GeDT is deterministic.

Proof. Consider any sum node v in a GeDT and assume, for simplicity, that it has two children u' and u''. Node v is associated with a partition $\{\boldsymbol{\mathcal{X}}_{u'}, \boldsymbol{\mathcal{X}}_{u''}\}$ of $\boldsymbol{\mathcal{X}}_v$. Any leaf ℓ which is a descendant of u', respectively u'', must have a support

which is a subset of $\boldsymbol{\mathcal{X}}_{u'}$, respectively $\boldsymbol{\mathcal{X}}_{u''}$. Assume that $p_{u'}(\mathbf{x}) > 0$ for certain $\mathbf{x}$, implying $\mathbf{x} \in \boldsymbol{\mathcal{X}}_{u'}$ and thus $\mathbf{x} \notin \boldsymbol{\mathcal{X}}_{u''}$. Therefore, $u''(\mathbf{x}) = 0$, since $\mathbf{x}$ is not in the support of any leaf below u''. The same argument holds for the reverse case and straightforwardly extends to arbitrarily many sum nodes. Thus v is deterministic. □

In Algorithm 3, we learn a density $p_\ell(\mathbf{x}, y)$ for each leaf ℓ, where we have not yet specified the model or learning algorithm. Thus, we denote GeDT(M) as a GeDT whose leaf densities are learnt by 'method M', where M might be graphical models, again PCs, or even neural-based density estimators (Kingma and Welling 2014; Rezende and Mohamed 2015). In order to ensure tractable marginalisation of the overall GeDT, however, we use either *fully factorised* leaves—for each leaf ℓ, $p_\ell(\mathbf{X}, Y) = p_\ell(X_1)p_\ell(X_2)\dots p_\ell(X_m)p_\ell(Y)$—or PCs learnt with *LearnSPN* (Gens and Domingos 2013). In both cases marginalisation at the leaves, and hence in the whole GeDT, is efficient. Regardless of the model M, we generally learn the leaves using the maximum likelihood principle, or some proxy of it. Thus, since the sum-weights are already set to the (global) maximum likelihood solution by Algorithm 3, the overall GeDT also fits the training data in the maximum-likelihood sense.

A key design choice is how to model the dependency between $\mathbf{X}$ and Y at the leaves: we might assume independence between them, i.e. assume $p_\ell(\mathbf{X}, Y) = p_\ell(\mathbf{X})p_\ell(Y)$ (*class-factorised* leaves)[3] or simply pass observations of both $\mathbf{X}$ and Y to a learning algorithm and let it determine the dependency structure itself (*full* leaves). Note that we are free to use different types of density estimators at different leaves in a single GeDT. Naturally, it makes sense to match the complexity of the estimator in a leaf to the number of samples it contains.

The main semantic difference between DTs and GeDTs is that a DT represents a classifier, i.e. a conditional distribution $f(\mathbf{x})$, while the corresponding GeDT represents a full joint distribution $p(\mathbf{X}, Y)$. The latter naturally lends itself towards classification by deriving the conditional distribution $p(Y \mid \mathbf{x}) \propto p(\mathbf{x}, Y)$. How are the original DT classifier $f(\mathbf{x})$ and the GeDT classifier $p(Y \mid \mathbf{x})$ related? In theory, $p(Y \mid \mathbf{x})$ might differ substantially from $f(\mathbf{x})$, since every feature might influence classification in a GeDT, even if it never appears in any decision node of the DT. In the case of class-factorised leaves, however, we obtain 'backwards compatibility'.

[3]Note that, as discussed in Chapter 5, such independence is only a context-specific one, conditional on the state of variables associated with sum nodes (Peharz et al. 2016; Poon and Domingos 2011). This assumption does not represent global independence between $\mathbf{X}$ and Y.

Theorem 5
Let f be a DT classifier and $p(Y \mid \mathbf{x})$ be a corresponding GeDT classifier, where each leaf in GeDT is class-factorised, i.e. of the form $p_\ell(Y)p_\ell(\mathbf{X})$, and where $p_\ell(Y)$ has been estimated in the maximum-likelihood sense. Then $f(\mathbf{x}) = p(Y \mid \mathbf{x})$, provided that $p(\mathbf{x}) > 0$.

Proof. Recall that the leaves in the GeDT are in one-to-one correspondence with the leaf cells $\boldsymbol{\mathcal{A}}$ of the DT, and that the support of any leaf is given by its corresponding $\mathcal{A} \in \boldsymbol{\mathcal{A}}$. Let $\ell_{\mathbf{x}}$ be the unique leaf in the GeDT whose cell is $\mathcal{A}(\mathbf{x})$. Since GeDTs are tree-shaped PCs containing only sum nodes, its joint distributions is either $p_{\ell_{\mathbf{x}}}(\mathbf{x}, y)$, for trivial GeDTs consisting only of $\ell_{\mathbf{x}}$, or can be written as

$$p(\mathbf{x}, y) = \sum_{v \in \mathrm{ch}(r)} w_{r,v} p_v(\mathbf{x}), \tag{6.2}$$

where r is the root node. Since a GeDT is deterministic, it has at most one non-zero child. From $p(\mathbf{x}) > 0$ it follows that the GeDT has *exactly* one non-zero child, say v', and Equation 6.2 can be written as $p(\mathbf{x}, y) = w_{r,v'} p_{v'}(\mathbf{x}, y)$. Now, since $p_{v'}(\mathbf{x}, y)$ is also a tree-shape PC containing only sums, it follows by induction that $p(\mathbf{x}, y) = \left(\prod_{(v,u)\in\Lambda} w_{v,u}\right) p_{\ell_{\mathbf{x}}}(\mathbf{x}, y)$, where Λ is the unique path from root to $\ell_{\mathbf{x}}$ following only non-zero nodes, and $w_{v,u}$ are the sum-weights of edges (v, u) in Λ. Since each leaf is class-factorised, we have $p_{\ell_{\mathbf{x}}}(\mathbf{x}, y) = p_{\ell_{\mathbf{x}}}(\mathbf{x}) p_{\ell_{\mathbf{x}}}(y)$, and $\left(\prod_{(v,u)\in\Lambda} w_{v,u}\right) p_{\ell_{\mathbf{x}}}(\mathbf{x}) p_{\ell_{\mathbf{x}}}(y) \propto p(y \mid \mathbf{x}) = p_{\ell_{\mathbf{x}}}(y) = f^{\mathcal{A}(\mathbf{x})}(\mathbf{x}) = f(\mathbf{x})$, since each $f^{\mathcal{A}}(\mathbf{x})$ is, like $p_{\ell_{\mathbf{x}}}$, learnt by the class proportions of samples in $\mathcal{A}$. □

Theorem 5 shows that DTs and GeDTs yield exactly the same classifier for class-factorised leaves and complete data. DTs achieve their most impressive performance when used as an ensemble in RFs. It is straightforward to convert each DT in an RF using Algorithm 3, yielding an ensemble of GeDTs. Such an ensemble, which we call a *Generative Forest* (GeF), computes the following conditional probability function

$$p(Y|\mathbf{X}) = \frac{1}{n_t} \sum_{j=1}^{n_t} p_j(Y|\mathbf{X}), \tag{6.3}$$

where n_t is the number of trees in the ensemble and p_j the density function defined by the jth GeDT. The result above naturally extends to ensembles, as clearly when all GeDTs in a GeF use class-factorised leaves, then according to

Theorem 5, GeFs yield exactly the same prediction function as their corresponding RFs. This means that the everyday practitioner can safely replace RFs with class-factorised GeFs, gaining the ability to classify under *missing input data*.

6.5 Handling Missing Values

The probably most frequent strategy to treat missing inputs in DTs and RFs is to use some *single imputation* technique, i.e. to first predict any missing values based on the observed ones, and then use the imputed sample as input to the classifier. A particularly prominent method is *K-nearest neighbour* (KNN) imputation, which typically works well in practice. This strategy, however, is not Bayes consistent and can in principle be arbitrarily bad. This can be shown with a simple example. Assume two multivariate Gaussian features X_1 and X_2 with $var(X_1) \geq \tau$, $var(X_2) \geq \tau$ for some $\tau > 0$, i.e. the variances of X_1 and X_2 are bounded from below. Let the conditional class distribution be $p(y \mid x_1, x_2) = \mathbb{1}(|x_2 - \mathbb{E}[X_2 \mid x_1]| > \epsilon)$, i.e. Y detects whether X_2 deviates more than ϵ from its mean, conditional on X_1. Assume X_2 is missing and use KNN to impute it, based on $X_1 = x_1$. KNN is known to be a consistent regressor, provided the number of neighbours goes to infinity but vanishes in comparison to the number of samples (Devroye et al. 1996). Thus, the imputation for X_2 based on x_1 converges to $\mathbb{E}[X_2 \mid x_1]$, yielding a constant prediction of $Y = 0$. It follows that by making ϵ arbitrarily small, we can push the classification error arbitrarily close to 1, while the true error goes to 0.

Assuming that inputs are missing at random (Little and Rubin 2019) and that we have only inputs $\mathbf{x}_o$ for some subset $\mathbf{X}_o \subset \mathbf{X}$, a GeDT naturally yields a classifier $p(y \mid \mathbf{X}_o)$ by marginalising missing features as in Equation 6.1. Recall that marginalisation in PCs, and thus in GeDTs, can be performed with a single feedforward pass, given that the GeDT's leaves permit efficient marginalisation. In our experiments, we use either fully factorised leaves or PC leaves learnt by LearnSPN (Gens and Domingos 2013), a prominent PC learner, such that we can efficiently and exactly evaluate $p(Y \mid \mathbf{X}_o)$ with a single pass through the network. Thus, a GeDT represents in fact $2^{|\mathbf{X}|}$ classifiers, one for each missingness pattern. Since the true data distribution yields Bayes optimal classifiers for each $\mathbf{X}_o$, and since the parameters of GeDTs are learnt in the maximum likelihood sense, using the GeDT predictor $p(y \mid \mathbf{X}_o)$ for missing data is natural. For a simplified variant of GeDTs, we can show that they converge to the true distribution and are therefore Bayes consistent classifiers for each $\mathbf{X}_o$. Theorem 6 assumes, without loss of generality, that all variables in $\mathbf{X}$ are continuous.

Theorem 6
Let $\mathbb{P}^*$ be an unknown data generating distribution with density $p^*(\mathbf{X}, Y)$, and let $\mathcal{D}_n$ be a dataset drawn i.i.d. from $\mathbb{P}^*$. Let $\mathcal{G}$ be a DT learnt with a DT learning algorithm, using axis-aligned splits. Let $\boldsymbol{\mathcal{A}}^n$ be the (rectangular) leaf cells produced by the learning algorithm. Assume it holds that i) $\lim_{n\to\infty} {}^{|\boldsymbol{\mathcal{A}}^n|\log(n)}/_n \to 0$ and ii) $\mathbb{P}^*(\{\mathbf{x} \mid \operatorname{diam}(\mathcal{A}_x^n) > \gamma\}) \to 0$ almost surely for all $\gamma > 0$, where $\operatorname{diam}(\mathcal{A})$ is the diameter of cell $\mathcal{A}$. Let $\mathcal{G}'$ be the GeDT corresponding to $\mathcal{G}$, obtained via Algorithm 3, where for each leaf ℓ, p_ℓ is of the form $p_\ell(Y)p_\ell(\mathbf{X})$, with $p_\ell(\mathbf{X})$ uniform on $\mathcal{A}_\ell$ and $p_\ell(Y)$ the maximum likelihood categorical distribution (fractions of class values of samples in $\mathcal{A}_\ell$). Then the GeDT distribution is l_1-consistent, i.e. $\sum_y \int |p(\mathbf{x}, y) - p^*(\mathbf{x}, y)| \mathrm{d}\mathbf{x} \to 0$, almost surely.

Before proving Theorem 6 we need to introduce some background. This theorem extends consistency results for collections of partitions of the state space $\boldsymbol{\mathcal{X}}$, as discussed by Lugosi and Nobel (Lugosi et al. 1996). A central notion is the *growth function* of such partitions.

Definition 5 (Growth function (Lugosi et al. 1996))
Let $\boldsymbol{\mathcal{X}}$ be some set and $\boldsymbol{\mathcal{F}}$ be a collection of finite partitions of $\boldsymbol{\mathcal{X}}$. Let $\boldsymbol{\xi} = \{\xi_1, \ldots, \xi_n\}$ be a set of points from $\boldsymbol{\mathcal{X}}$. Let $\Delta(\boldsymbol{\mathcal{F}}, \boldsymbol{\xi})$ be the number of distinct partitions induced by $\boldsymbol{\mathcal{F}}$, that is the size of set $\{\{\boldsymbol{\xi} \cap \mathcal{A} \mid \mathcal{A} \in \boldsymbol{\mathcal{A}}\} \mid \boldsymbol{\mathcal{A}} \in \boldsymbol{\mathcal{F}}\}$. The growth function is defined as $\Delta^*(\boldsymbol{\mathcal{F}}) = \sup_{\boldsymbol{\xi}} \Delta(\boldsymbol{\mathcal{F}}, \boldsymbol{\xi})$, where the $\sup$ ranges over all sets of n points from $\boldsymbol{\mathcal{X}}$.

Note that the growth function Δ^* is defined akin to the *dichotomic growth function*, as introduced by Vapnik and Chervonenkis and well known in statistical learning theory (Vapnik 1998). In particular, we derive the following bound of Δ^*.

Proposition 2
Let $\boldsymbol{\mathcal{X}}$ be some set and $\boldsymbol{\mathcal{C}}$ be any collection of subsets of $\boldsymbol{\mathcal{X}}$. Let $\Phi(\boldsymbol{\mathcal{C}}, \boldsymbol{\xi})$ be the shatter coefficient of point set $\boldsymbol{\xi}$ and $\Phi^*(\boldsymbol{\mathcal{C}}) = \sup_{\boldsymbol{\xi}} \Phi(\boldsymbol{\mathcal{C}}, \boldsymbol{\xi})$ be the dichotomic growth function (Vapnik 1998). Let $\boldsymbol{\mathcal{F}}$ be a collection of finite partitions of $\boldsymbol{\mathcal{X}}$, as in Definition 5, where the maximal partition size is $J := \sup_{\boldsymbol{\mathcal{A}} \in \boldsymbol{\mathcal{F}}} |\boldsymbol{\mathcal{A}}|$. If $\boldsymbol{\mathcal{C}} = \{\mathcal{A} \mid \mathcal{A} \in \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \in \boldsymbol{\mathcal{F}}\}$ then

$$\Delta(\boldsymbol{\mathcal{F}}, \boldsymbol{\xi}) \leq \Phi(\boldsymbol{\mathcal{C}}, \boldsymbol{\xi})^J, \tag{6.4}$$

and moreover $\Delta^*(\boldsymbol{\mathcal{F}}) \leq \Phi^*(\boldsymbol{\mathcal{C}})^J$.

Proof. Let the point set $\boldsymbol{\xi}$ be fixed. Any partition $\{\boldsymbol{\xi} \cap \mathcal{A} \mid \mathcal{A} \in \boldsymbol{\mathcal{A}}\}$, for some $\boldsymbol{\mathcal{A}} \in \boldsymbol{\mathcal{F}}$, can be written as $\{\boldsymbol{\xi} \cap \mathcal{A}_1, \ldots, \boldsymbol{\xi} \cap \mathcal{A}_J\}$ for some $\mathcal{A}_1, \ldots, \mathcal{A}_J \in \boldsymbol{\mathcal{C}}$, since $\boldsymbol{\mathcal{C}}$ contains all cells which appear in $\boldsymbol{\mathcal{F}}$. Thus, $\Delta(\boldsymbol{\mathcal{F}}, \boldsymbol{\xi}) \leq |\{\{\boldsymbol{\xi} \cap \mathcal{A}_1, \ldots, \boldsymbol{\xi} \cap \mathcal{A}_J\} \mid \mathcal{A}_1, \ldots, \mathcal{A}_J \in \boldsymbol{\mathcal{C}}\}|$. Note that the number of partitions of this form is bounded by

$$|\{\{\boldsymbol{\xi} \cap \mathcal{A}_1, \ldots, \boldsymbol{\xi} \cap \mathcal{A}_J\} \mid \mathcal{A}_1, \ldots, \mathcal{A}_J \in \boldsymbol{\mathcal{C}}\}| \leq \bigtimes_{j=1}^{J} |\{\boldsymbol{\xi} \cap \mathcal{A}_j \mid \mathcal{A}_j \in \boldsymbol{\mathcal{C}}\}|. \tag{6.5}$$

The right hand side of Equation 6.5 is $\Phi(\boldsymbol{\mathcal{C}}, \boldsymbol{\xi})^J$, and thus Equation 6.4 follows. $\Delta^*(\boldsymbol{\mathcal{F}}) \leq \Phi^*(\boldsymbol{\mathcal{C}})^J$ follows from applying $\sup_{\boldsymbol{\xi}}$ on both sides of Equation 6.4. □

In our case, we study partitions $\boldsymbol{\mathcal{A}}$ induced by a DT, each of which divides $\boldsymbol{\mathcal{X}}$ into a set of hyper-rectangles.[4] Hence, we consider the collection of partitions $\boldsymbol{\mathcal{F}}$ containing all possible partitions whose sets are hyper-rectangles. We are now ready to prove Theorem 6.

Proof. Let $\boldsymbol{\mathcal{F}}^n$ be the collection of all DT partitions which can be generated for sample size n, i.e. $\boldsymbol{\mathcal{A}}^n \in \boldsymbol{\mathcal{F}}^n$. By Proposition 2, we know that $\Delta^*(\boldsymbol{\mathcal{F}}^n) \leq \Phi^*(\boldsymbol{\mathcal{C}})^{|\boldsymbol{\mathcal{A}}^n|}$, where $\boldsymbol{\mathcal{C}}$ is the collection of all sub-rectangles in $\boldsymbol{\mathcal{X}}$. The VC dimension (Vapnik 1998) of $\boldsymbol{\mathcal{C}}$ is known to be $2|\mathbf{X}|$, and consequently, by Sauer's lemma, $\Delta^*(\boldsymbol{\mathcal{F}}) \leq \Phi^*(\boldsymbol{\mathcal{C}})^{|\boldsymbol{\mathcal{A}}^n|} \leq Cn^{2|\boldsymbol{\mathcal{A}}^n||\mathbf{X}|}$, where C is a constant depending only on $|\mathbf{X}|$. Therefore, if condition i) holds ($\lim_{n\to\infty} {}^{|\boldsymbol{\mathcal{A}}^n|\log(n)}/_{n} \to 0$) it follows that $\frac{\log \Delta^*}{n} \to 0$. Thus, together with condition ii) all conditions of Theorems 1 and 2 in (Lugosi et al. 1996) hold.

Since the GeDT is deterministic, its distribution can be written as

$$p(\mathbf{x}, y) = \left(\prod_{(v,u)\in\Lambda} w_{v,u} \right) p_{\ell_{\mathbf{x}}}(\mathbf{x}, y), \tag{6.6}$$

where $\ell_{\mathbf{x}}$ is the unique non-zero leaf in the GeDT, Λ is the unique path from the root to $\ell_{\mathbf{x}}$ following only non-zero nodes, and $w_{v,u}$ are the sum-weights of edges (v, u) in Λ (see also proof of Theorem 5).

It is easy to see that $\prod_{(v,u)\in\Lambda} w_{v,u} = \hat{\mathbb{P}}(\mathcal{A}_{\mathbf{x}})$, where $\hat{\mathbb{P}}$ is the empirical distribution of $\mathcal{D}_n$, i.e. the fraction of data points falling in $\mathcal{A}_{\mathbf{x}}$ (see Algorithm 1). The

[4]Here, we assume for simplicity that all variables are continuous. Including discrete variables with finitely many states can be done by applying similar arguments to each of the finitely many joint states.

distribution computed by each leaf ℓ is, by assumption, $p_\ell(\mathbf{x}, y) = p_\ell(y)\frac{1}{\mathrm{vol}(\mathcal{A}_\mathbf{x})}$, where $\mathrm{vol}(\mathcal{A})$ is the volume (Lebesgue measure) of $\mathcal{A}$. Thus, we can write Equation 6.6 as

$$p(\mathbf{x}, y) = p_\ell(y)\hat{\mathbb{P}}(\mathcal{A}_\mathbf{x})\frac{1}{\mathrm{vol}(\mathcal{A}_\mathbf{x})}.$$

By Theorem 1 in (Lugosi et al. 1996), $\hat{\mathbb{P}}(\mathcal{A}_\mathbf{x})\frac{1}{\mathrm{vol}(\mathcal{A}_\mathbf{x})}$ converges to $p^*(\mathbf{x})$, while by Theorem 2 in (Lugosi et al. 1996), $p_\ell(y)$ converges to $p^*(y \,|\, \mathbf{x})$, both in $l1$-sense. Clearly both factors, $\hat{\mathbb{P}}(\mathcal{A}_\mathbf{x})\frac{1}{\mathrm{vol}(\mathcal{A}_\mathbf{x})}$ and $p_\ell(y)$, have bounded $l1$-norm. Thus, their product converges to $p^*(y \,|\, \mathbf{x})p^*(\mathbf{x}) = p^*(y, \mathbf{x})$, which concludes the proof. □

Note that the assumptions in Theorem 6 are in line with consistency results for DTs. See for example (Breiman et al. 1984; Devroye et al. 1996; Lugosi et al. 1996), all of which require, in some sense, that the number of cells vanishes in comparison to the number of samples, and that the cell sizes shrink to zero. Theorem 6 naturally leads to the Bayes-consistency of GeDTs and GeFs under missing inputs.

Corollary 3
Under assumptions of Theorem 6, any GeDT predictor $p(Y \,|\, \mathbf{X}_o)$, for $\mathbf{X}_o \subseteq \mathbf{X}$ is Bayes consistent.

Proof. Since $p(y, \mathbf{x})$ converges almost surely to $p^*(y, \mathbf{x})$ in $l1$-sense, it gives rise to the Bayes optimal classifier $\arg\max_y p^*(y, \mathbf{x})$. Consider any $X_i \in \mathbf{X}$. The marginal distribution, X_i marginalised out, is $\int p(y, \mathbf{x}_{\neg i}, x_i)\mathrm{d}x_i$. Since

$$\begin{aligned}\int |p(y, \mathbf{x}_{\neg i}) - p^*(y, \mathbf{x}_{\neg i})|\mathrm{d}\mathbf{x}_{\neg i} \\ = \int \left| \int p(y, \mathbf{x}_{\neg i}, x_i) - p^*(y, \mathbf{x}_{\neg i}, x_i)\mathrm{d}x_i \right| \mathrm{d}\mathbf{x}_{\neg i} \\ \leq \int |p(y, \mathbf{x}) - p^*(y, \mathbf{x})| d\mathbf{x},\end{aligned}$$

also the marginal converges in $l1$-sense to the true $p^*(y, \mathbf{x}_{\neg i})$. By repeating the argument, every sub-marginal converges, and thus gives rise to the corresponding Bayes optimal classifier. □

Corollary 4
Assume a GeF whose GeDTs are learnt under assumptions of Theorem 6. Then the GeF of GeDT predictors $p(Y \,|\, \mathbf{X}_o)$, for any $\mathbf{X}_o \subseteq \mathbf{X}$, is Bayes consistent.

Proof. This follows directly from Proposition 1 in (Biau et al. 2008), whereby if a sequence of classifiers is Bayes-consistent, then the classifier obtained by averaging them is also consistent. □

6.6 Computational Complexity

Let n be the total number of samples and m the total number of features. Regarding the learning algorithm, a Random Forest and its corresponding PC only differ in the distributions at leaves, which use a partition of the data. Therefore, assuming a tree is grown as in (Breiman 2001) with $\lceil m/c \rceil$ features considered at each split (c a positive natural), structure learning in both models has worst-case asymptotic complexity of $\mathcal{O}(mr\ n \log n)$, where $r \in \mathcal{O}(n)$ is the number of internal nodes in the obtained tree (Louppe 2014). For GeDTs, however, there is the additional cost of learning a distribution at each leaf. If $q(m)$ is the worst-case cost of the leaf learner for a constant amount of data, then the overall time complexity (for learning all leaves) is $\mathcal{O}(r\ q(m))$.

Nonetheless, if the leaf learner is such that $q(m) \leq \mathcal{O}(mn \log n)$, then the complexity is dominated by the structure learning and Random Forests and GeFs have the same worst-case asymptotic complexity of $\mathcal{O}(n_t\ (mr\ n \log n + rq(m))) \leq \mathcal{O}(n_t\ mr\ n \log n)$, where n_t is the number of trees in the model. Note that $q(m) \leq \mathcal{O}(mn \log n)$ holds for many learning algorithms when only a small number of training samples fall in each leaf—namely, LearnSPN and fully-factorised leaves—provided the reasonable assumption that m is $\mathcal{O}(n)$.

To perform inference for a complete test sample, GeDTs require traversing the whole structure once (hence time $\mathcal{O}(r)$), while DTs have a worst-case of $\mathcal{O}(d)$, where d is the height of the tree. However, we can bring the complexity of GeDTs down to $\mathcal{O}(d)$ by placing the indicators that define the decisions of the internal nodes of the DT near the corresponding internal nodes of the GeDT. This requires augmenting GeDTs with product nodes, one for each internal sum node. Every new product node has two children: a sum node and an indicator mimicking the decision tree split, that is, the indicator only evaluates to one if that path in the tree is active. Figure 6.2 illustrates this idea using our running example, where the densities are as follows

$$
\begin{aligned}
p_1(X_1, X_2, Y) &= p_1(X_1, X_2)(0 \cdot \mathbb{1}(Y = 0) + 1 \cdot \mathbb{1}(Y = 1)), \\
p_2(X_1, X_2, Y) &= p_2(X_1, X_2)(0.25 \cdot \mathbb{1}(Y = 0) + 0.75 \cdot \mathbb{1}(Y = 1)), \\
p_3(X_1, X_2, Y) &= p_3(X_1, X_2)(1 \cdot \mathbb{1}(Y = 0) + 0 \cdot \mathbb{1}(Y = 1)).
\end{aligned}
$$

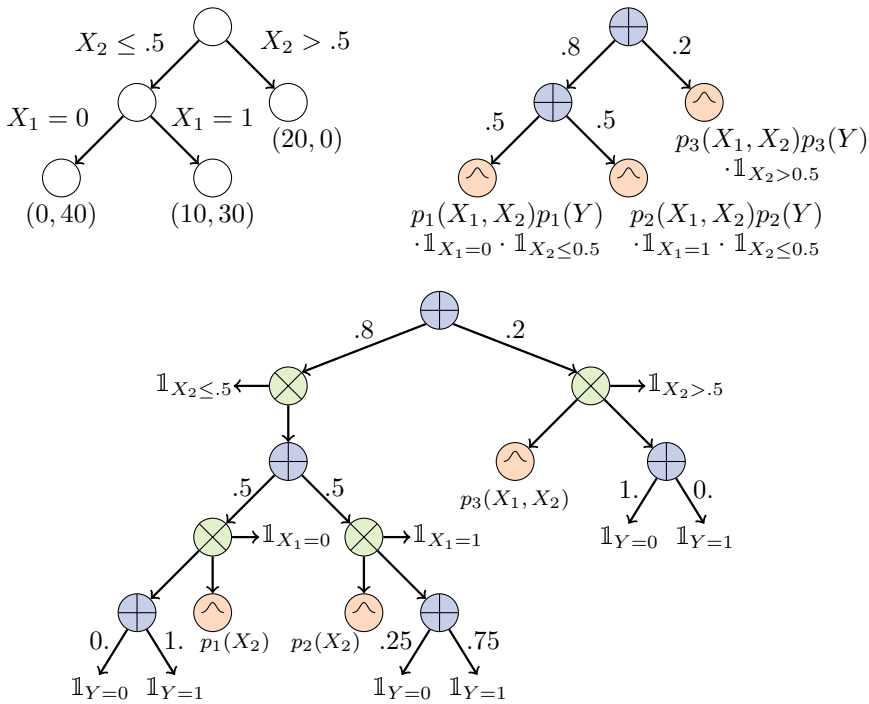

Figure 6.2: Illustration of 'pulling indicators up' to speed up computations in a Generative Tree (in the example, $\mathbf{X}$ and Y factorise at the leaves). The original decision tree (DT) is shown on the top left. The other two graphs represent the Probabilistic Circuit corresponding to the original DT and encode the very same distribution, even if the one in the bottom is not decomposable.

This idea does not change results, since it is the same as bringing the common indicators that appeared in the leaves of a sub-tree up towards the root of that sub-tree using the distributive property of multiplication (the lack of decomposability is tackled by the determinism of the indicators). By evaluating indicators as soon as possible in a top-down recursive computation, we can avoid computing all sub-trees for which a zero is returned to a product node. With this type of computational graph, GeFs and RFs have a similar inference procedure. Predicting the class of an instance amounts to traversing each tree and evaluating

the corresponding leaf, and thus the inference complexity is $\mathcal{O}(n_t d)$.

For incomplete data, however, GeDTs need to reach every active leaf (just as Friedman's method). Assuming the number of missing values in each instance is bounded by a constant, GeFs still take time $\mathcal{O}(n_t d)$, being faster than Random Forests with KNN imputation, which in the worst case take time $\mathcal{O}(n_t d + nm)$. For large (non-constant) percentages of missing values, GeFs can be as slow as $\mathcal{O}(n_t r)$ (as it may need to reach all leaves). In this case of large numbers of missing values per instance, GeFs are faster than Random Forests with KNN imputation if $d \approx r$ but slower if $d \ll r$.

6.7 Regression Tasks

In theory, we can apply the same methods to regression tasks and still be able to handle missing values and do outlier detection. However, this requires extra care on how we represent and propagate the conditional distribution $p(Y \mid \mathbf{X})$ through the network. To see this, consider the case where a split variable is not observed. Inference in a GeDT will boil down to evaluating a mixture of at least two (possibly more depending on how many times the split variable is used) leaf distributions over Y. For classification, the computation and propagation of such mixture is trivial since for each leaf node ℓ, $p_\ell(Y \mid \mathbf{X})$ is a categorical distribution, and a mixture of categorical distributions results again in a categorical distribution. No matter how many leaves we need to aggregate during inference with missing data, the representation of the distribution over Y has constant size, and we can easily propagate it through the graph of a GeDT.

That is not true in general for continuous distributions. Even in the simple case where we model each $p_\ell(Y \mid \mathbf{X})$ as a normal distribution, a mixture of normal distributions can no longer be represented as a single normal distribution, and the propagation through the network becomes costlier the more leaves we need to aggregate. This could be tackled, for example, with empirical distributions (histograms) in which case we could keep the representation size fixed at the cost of some approximation error. It would also be possible to do a binary search for the most likely y value, which would require a number of passes through the network. At any rate, we leave the extension of GeDTs to regression tasks for future work and concentrate our experiments on classification tasks.

6.8 Experiments

We run a series of classification tasks with incomplete data to compare our models against surrogate splits (Breiman et al. 1984; Therneau et al. 1997), Friedman's method (Friedman 1977; Quinlan 1987a), and mean (mode), KNN ($k = 7$) and MissForest (Stekhoven and Bühlmann 2012) imputation. In particular, we experiment with two variants of GeFs: one with fully-factorised leaves, which we denote simply GeF, and another with leaves learnt via LearnSPN (Gens and Domingos 2013), which we call GeF(LearnSPN). In particular, we use a transformation of GeFs into a clever PC that prunes unnecessary sub-trees (Correia and de Campos 2019), speeding up computations and achieving time complexity comparable to the original DTs and RFs (see Section 6.6).

In all experiments, GeF, GeF(LearnSPN) and the RF share the exact same structure (partition over the feature space) and are composed of 100 trees; including more trees has been shown to yield only marginal gains in most cases (Probst and Boulesteix 2018). In GeF(LearnSPN), we run LearnSPN only for leaves with more than 30 samples, defaulting to a fully factorised model in smaller leaves. More information about the experimental setup is given in Appendix A, while more extensive experimental results, detailing the performance of GeFs and competing methods in each dataset, are available in the appendix.

Table 6.1: Accuracy at 30% percent of missing values at test time with 95% confidence intervals. The best performing model is underlined, whereas all models within its confidence interval appear in bold.

Dataset	n	Surrogate	Friedman	Mean	KNN	MissForest	GeF	GeF(LSPN)
dresses	500	45.48 ± 1.57	55.8 ± 1.21	**58.18** ± .85	56.62 ± 1.57	55.68 ± .95	57.12 ± 1.11	57.14 ± 1.07
wdbc	569	94.96 ± .36	94.96 ± .35	94.92 ± .58	95.59 ± .41	94.92 ± .36	95.64 ± .47	**96.26** ± .42
diabetes	768	72.97 ± .73	**73.35** ± .70	71.67 ± .84	72.4 ± .75	72.46 ± .92	**73.93** ± .63	**73.83** ± .72
vehicle	846	**71.61** ± .92	67.12 ± .79	63.27 ± 1.05	**71.77** ± 1.01	70.69 ± 1.33	**72.39** ± 1.13	**72.77** ± 1.27
vowel	990	78.79 ± .86	70.81 ± 1.45	64.51 ± .83	85.62 ± .66	81.85 ± 1.10	**89.25** ± .77	**89.59** ± .91
credit-g	1000	71.97 ± .32	72.42 ± .31	73.01 ± .64	73.06 ± .65	73.03 ± .78	**73.81** ± .38	**73.97** ± .36
mice	1080	95.84 ± .44	91.01 ± .77	84.91 ± 1.03	97.7 ± .50	96.08 ± .52	98.38 ± .32	**99.06** ± .15
authent.	1372	88.65 ± .74	87.13 ± .69	84.47 ± .79	**91.98** ± .55	90.74 ± .37	90.33 ± .65	89.66 ± .64
cmc	1473	48.7 ± .79	**49.8** ± .38	47.67 ± .86	48.38 ± .58	48.28 ± .90	**49.96** ± 1.03	**50.08** ± 1.05
segment	2310	93.32 ± .27	84.14 ± .69	78.34 ± .68	**94.25** ± .32	93.21 ± .41	93.42 ± .20	93.41 ± .34
dna	3186	**90.53** ± .31	77.23 ± .36	83.91 ± .43	89.31 ± .33	**90.76** ± .34	87.42 ± .19	82.99 ± .25
splice	3190	86.09 ± .46	84.76 ± .65	84.65 ± .28	89.06 ± .53	86.18 ± .26	**91.1** ± .50	85.69 ± .53
krvskp	3196	73.62 ± .92	82.81 ± .73	83.58 ± .64	86.58 ± .43	86.24 ± .56	**88.35** ± .39	**88.65** ± .32
robot	5456	91.74 ± .23	84.73 ± .57	89.39 ± .28	92.74 ± .25	91.72 ± .37	92.97 ± .28	**94.67** ± .20
texture	5500	95.31 ± .15	89.85 ± .34	84.24 ± .42	**97.13** ± .15	95.4 ± .17	95.93 ± .15	**97.12** ± .13
wine	6497	84.49 ± .17	82.45 ± .10	83.2 ± .15	85.73 ± .26	**85.95** ± .12	85.22 ± .18	**85.85** ± .19
gesture	9873	58.37 ± .15	52.86 ± .27	55.41 ± .21	**61.62** ± .22	**61.48** ± .26	58.65 ± .18	60.2 ± .21
phishing	11055	81.52 ± .50	88.98 ± .17	88.02 ± .19	92.06 ± .13	91.18 ± .18	92.99 ± .08	**93.3** ± .06
bank	41188	90.42 ± .18	90.3 ± .15	90.09 ± .11	**90.64** ± .14	90.4 ± .19	**90.79** ± .21	**90.77** ± .21
jungle	44819	63.45 ± .24	71.91 ± .12	66.89 ± .40	66.25 ± .15	65.67 ± .20	**72.4** ± .12	**72.3** ± .11
electricity	45312	79.79 ± .09	77.47 ± .10	73.24 ± .19	80.55 ± .11	81.21 ± .10	82.23 ± .12	**82.64** ± .11

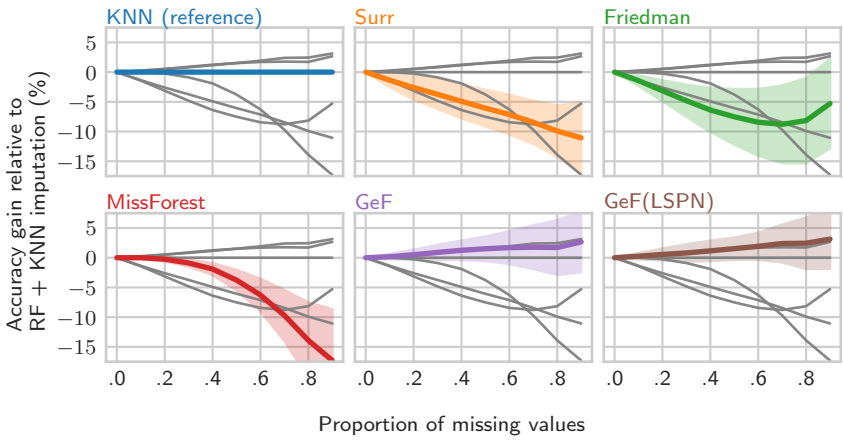

Figure 6.3: Average accuracy gain relative to RFs (100 trees) plus KNN imputation against proportion of missing values. The same plot is repeated six times, each time highlighting one method. The average as well as the confidence intervals (95%) are computed across the 21 datasets of Table 6.1.

We compare the accuracy of the methods in a selection of datasets from the OpenML-CC18 benchmark[5] (Vanschoren et al. 2013) and the wine-quality dataset (Moro et al. 2011). Table 6.1 presents results for 30% of missing values at test time (different percentages are shown in the appendix), with 95% confidence intervals across 10 repetitions of 5-fold cross-validation. GeF models outperform other methods in almost all datasets, validating that the joint distributions at the leaves provide enough information for computing the marginalisation in Equation 6.1. We also note that increasing the expressive power of the models at the leaves seems worthwhile, as GeF(LSPN) outperforms the vanilla GeF in about half of the datasets.

Similar conclusions are supported by Figure 6.3, where we plot the average gain in accuracy relative to RF + KNN imputation at different proportions of missing values. While earlier built-in methods, Friedman's and surrogate splits, perform poorly (justifying the popularity of imputation techniques for RFs), GeFs are on average more than 3% more accurate than KNN imputation.

[5] `https://www.openml.org/s/99/data`

For the sake of space, a thorough exposition of these experiments is deferred to the appendix, where we fully describe the experimental procedure, show different percentages of missing data and include results with PCs learnt via *class-selective* LearnSPN (Correia and de Campos 2019), as baseline for a standard generative model.

6.9 Conclusion

By establishing a connection between Decision Trees (DTs) and Probabilistic Circuits (PCs), we have upgraded DTs to a full joint model over both inputs and outputs, yielding their generative counterparts, called GeDTs. The fact that GeDTs, and their ensemble version GeFs, are 'backwards compatible' to DTs and RFs, while offering benefits like consistent classification under missing inputs and outlier detection, makes it easy to adopt them in everyday practice. Missing data and outliers, however, are just the beginning. We believe that many of the current challenges in machine learning, like explainability, interpretability, and (adversarial) robustness are but symptoms of an overemphasis on purely discriminative methods in the past decades, and that hybrid generative approaches—like the one in this chapter—will contribute significantly towards mastering these current challenges.

Chapter 7

Robustness of Probabilistic Circuits

Probabilistic Circuits (PCs) are machine learning models that combine good representational power with tractable inference. Like any standard machine learning model, they are not immune to overfitting—particularly true for PCs with large, deep architectures. PCs do, however, have the benefit of principled probabilistic semantics that facilitate the analysis of the reliability of its predictions. In this chapter, we explore credal methods that allow us to represent an entire set of PCs and study how predictions vary within such a set. That is captured by the concept of ϵ-robustness, a per-instance measure that indicates how much we would have to perturb the parameters of a PC to change its prediction when evaluated on a given instance. As we demonstrate with LearnSPN and Generative Forests, ϵ-robustness is highly correlated with accuracy and particularly useful in ensemble modelling. Unfortunately, evaluating ϵ-robustness can be costly for arbitrary PC architectures. Therefore, we introduce class-selective architectures, which are well suited for classification tasks and enable efficient robustness computation.

This chapter is based on **Alvaro Correia** and Cassio de Campos: Towards Scalable and Robust Sum-Product Networks, *International Conference on Scalable Uncertainty Management, Springer, Cham,* (2019); and **Alvaro Correia**, Robert Peharz and Cassio de Campos: Towards Robust Classification with Deep Generative Forests, *ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning* (2020).

7.1 Introduction

As discussed in previous chapters, Probabilistic Circuits (or PCs) are a class of deep probabilistic graphical models where exact marginal inference is always tractable. Still, PCs can capture high tree-width models (Poon and Domingos 2011) and are capable of representing complex and highly multidimensional distributions (Delalleau and Bengio 2011). This promising combination of efficiency and representational power has motivated several applications of PCs to a variety of machine learning tasks (Amer and Todorovic 2016; Cheng et al. 2014; Nath and Domingos 2016; Pronobis and Rao 2017; Sguerra and Cozman 2016; Wang and Wang 2018).

Yet, as any other machine learning model, PCs learnt from data are prone to overfitting when evaluated at poorly represented regions of the feature space, leading to overconfident and often unreliable conclusions. However, due to the probabilistic semantics of PCs, we can mitigate that issue through a principled analyses of the reliability of each output. A notable example is Credal PCs (CPCs)[1](Mauá et al. 2017), an extension of PCs to imprecise probabilities where we can compute a measure of the robustness of each prediction. Such robustness values are useful tools for decision-making, as they are highly correlated with accuracy, and thus tell us when to trust the prediction of the model: if the robustness of a prediction is low, we can suspend judgement or even resort to another machine learning model.

Unfortunately, computing robustness requires many passes through the network, which limits the scalability of credal Probabilistic Circuits. We address that by introducing class-selective (C)PCs, a type of architecture that enables efficient robustness computations in classification tasks due to their independent sub-networks: one for each label in the data. Class-selective (C)PCs not only enable fast robustness estimation but also outperform general (C)PCs in classification tasks. In our experiments, their accuracy was comparable to that of state-of-the-art methods, such as XGBoost (Chen et al. 2016).

[1]Most of the work presented in this chapter (Correia and de Campos 2019) was developed in the context of Sum-Product Networks (or SPNs), a type of Probabilistic Circuit satisfying smoothness and decomposability. However, for the sake of congruence with the rest of the thesis we will favour the more general denomination Probabilistic Circuits or PCs.

7.2 (Credal) Probabilistic Circuits

Before giving a formal definition of (C)PCs, we introduce the necessary notation and background. As in previous chapters, we have set of m random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$ each assuming values in some compact set $\mathcal{X}_i$. Since we are primarily concerned with classification tasks in this chapter, we reserve Y to denote the target variable with state space $\mathcal{Y}$. As usual we denote the realisation of (a set of) random variables in lowercase letters and use boldface to distinguish vectors from single values (e.g. $\mathbf{X} = \mathbf{x}$, $X_i = x_i$). Taking full advantage of tractable marginals in PCs, the results herein are valid for arbitrary subsets of $\mathbf{X}$. Therefore, when only a subset of the variables is concerned, we use an indexing set $\mathcal{E}$ to identify the corresponding variables $\mathbf{X}_\mathcal{E} = \{X_i : i \in \mathcal{E}\}$ and their realisations $\mathbf{x}_\mathcal{E}$. Here $\mathbf{x}_\mathcal{E}$ is what we call *partial evidence*, as not every variable is observed.

As in other chapters, we often need to refer to specific nodes in the graph. We reserve letters u and v for individual nodes, and r for the root node. We denote the set of children of a node v as $\mathrm{ch}(v)$. We represent the density function defined by a PC with parameters $\boldsymbol{\theta}$, structure $\mathcal{G}$ and root r as $p_r(\cdot \mid \boldsymbol{\theta}, \mathcal{G})$. Similarly, we use $p_v(\cdot \mid \boldsymbol{\theta}_v, \mathcal{G}_v)$ to denote the density function defined by any node v in the graph, where $\boldsymbol{\theta}_v$ and $\mathcal{G}_v$ are, respectively, the parameters and the structure of the PC rooted at node v. However, since we are primarily concerned with variations in $\boldsymbol{\theta}$, we shall often omit the dependence on the graph $\mathcal{G}$ and use simply $p_v(\cdot \mid \boldsymbol{\theta}_v)$ instead. At times, we shall also omit the node from the notation when it is clear from context, for example using only $p(\cdot \mid \boldsymbol{\theta})$ for the density computed at the root node. We also need to identify the weights of a sum node. Every arc from a sum node v to a child u is associated with a non-negative weight $w_{v,u}$ such that $\sum_{u \in \mathrm{ch}(v)} w_{v,u} = 1$, i.e. each sum node has normalised weights $\boldsymbol{w}_v$, where $\boldsymbol{w}_v$ is a vector collecting all weights $w_{v,u}$ associated with arcs from v to children u. Constraining the weights of sum nodes to be normalised does not affect the generality of the model (Peharz et al. 2015).

A Credal PC (CPC) (Mauá et al. 2017) is defined similarly, except for containing sets of weight vectors in each sum node instead of a single weight vector. More precisely, a CPC C is defined by a set of PCs $C = \{p(\cdot \mid \boldsymbol{\theta}, \mathcal{G}) : \boldsymbol{\theta} \in \mathcal{C}\}$ over the same graph $\mathcal{G}$, where $\mathcal{C}$ is the Cartesian product of finitely-generated simplexes $\mathcal{C}_v$, one for each sum node v, such that the weights $\boldsymbol{w}_v$ of a sum node v are constrained by $\mathcal{C}_v$. We formalise this concept with ϵ-contaminated sets. If $\boldsymbol{w}$ is the vector of weights of a given sum node, then for some $0 \leq \epsilon \leq 1$, its

ϵ-contamination is given by the set

$$\mathcal{C}_v = \{(1-\epsilon)\boldsymbol{w}_v + \epsilon\boldsymbol{v} : v_j \geq 0, \sum_j v_j = 1\}, \tag{7.1}$$

where $\boldsymbol{v}$ is a vector of same size as $\boldsymbol{w}_v$.

Such ϵ-contaminated sets are a straightforward way to define the simplex of each sum node. A credal PC is typically obtained by setting a single ϵ value for the entire network and defining the simplex associated with each sum node $\mathcal{C}_v$ as in Equation 7.1 (Maua et al. 2018). The experiments in this chapter also rely on ϵ-contaminated PCs, however, the theoretical results, unless otherwise stated, apply to polytopes $\mathcal{C}_v$ of any form.

While a PC represents one joint distribution over its variables, a CPC represents a set of joint distributions. Therefore, one might be interested in bounding the (log-)likelihood of some evidence $\mathbf{x}_\mathcal{E}$, i.e. computing $\min_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$ and $\max_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$. Fortunately, for PCs with tree structures and bounded number of children in each sum node, this can be computed efficiently with the following set of equations

$$\min_{\boldsymbol{\theta}} p_v(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta}) = \begin{cases} \min_{\boldsymbol{\theta}} p_v(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta}) & \text{if } v \text{ a dist. node,} \\ \prod_{u \in \mathrm{ch}(v)} \min_{\boldsymbol{\theta}_u} p_u(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta}_u) & \text{if } v \text{ a prod. node,} \\ \min_{\boldsymbol{w}_v} \sum_{u \in \mathrm{ch}(v)} w_{v,u} \left(\min_{\boldsymbol{\theta}_u} p_u(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta}_u)\right) & \text{if } v \text{ a sum node.} \end{cases}$$

Naturally, an analogous procedure applies to $\max_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$. That tells us that finding $\min_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$ is similar to performing inference in a regular PC but with the additional cost of solving a small linear program at each sum node. Intuitively, if the PC structure is a tree, the sub-networks of each child of a node v do not share any weights, $\bigcap_{u \in \mathrm{ch}(v)} \boldsymbol{\theta}_u = \varnothing$, and can be optimised local and independently. We can then start at distribution nodes, which typically compute simple univariate distributions for which minimum (resp. maximum) values are easily obtained, and propagate up through the network only the minimum (resp. maximum) values of each node. This result is formalised below, where Corollary 5 is a small variation of the result in (Maua et al. 2018).

Theorem 7 (Theorem 1 in (Maua et al. 2018))
Consider a CPC $C = \{p(\cdot \mid \boldsymbol{\theta}, \mathcal{G}) : \boldsymbol{\theta} \in \mathcal{C}\}$. Computing $\min_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$ and $\max_{\boldsymbol{\theta}} p(\mathbf{x}_\mathcal{E} \mid \boldsymbol{\theta})$ takes $O(|\boldsymbol{\theta}| \cdot L)$ time, where $|\boldsymbol{\theta}|$ is the number of parameters, and L is an upper bound on the cost of solving a linear program of the form $\min_{\boldsymbol{w}_v} \sum_{u \in \mathrm{ch}(v)} c_{v,u} w_{v,u}$ subject to $\boldsymbol{w}_v \in \mathcal{C}_v$.

Corollary 5
Consider a CPC $C = \{p(\cdot \mid \boldsymbol{\theta}, \mathcal{G}) : \boldsymbol{\theta} \in \mathcal{C}\}$ with bounded number of children per sum node and specified by simplexes $\mathcal{C}_v$ of (finitely many) constraints of the form $lb_{v,u} \leq w_{v,u} \leq ub_{v,u}$ for given rationals $lb_{v,u} \leq ub_{v,u}$. Computing $\min_{\boldsymbol{\theta}} p(\mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})$ and $\max_{\boldsymbol{\theta}} p(\mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})$ can be solved in time $O(|\,\boldsymbol{\theta}\,|)$.

Proof. When local simplexes $\mathcal{C}_v$ have constraints $\mathrm{lb}_{v,u} \leq w_{v,u} \leq \mathrm{ub}_{v,u}$, then the local optimisations $p_v(\mathbf{x} \mid \boldsymbol{\theta}_v) = \min_{\boldsymbol{w}_v} \sum_{u \in \mathrm{ch}(v)} w_{v,u} p_u(\mathbf{x} \mid \boldsymbol{\theta}_u)$ are equivalent to fractional knapsack problems (Korte and Vygen 2012), which are solvable in constant time for nodes with bounded number of children. Thus, the overall running time is $O(|\,\boldsymbol{\theta}\,|)$. □

An important consequence of the results above is that one can use CPCs to obtain lower and upper bounds $\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(f|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}})$ and $\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(f|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}})$ on the expected value of some function f of a variable, conditioned on evidence $\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}$. Each choice of weights $\boldsymbol{\theta}$ of a CPC $C = \{p(\cdot \mid \boldsymbol{\theta}, \mathcal{G}) : \boldsymbol{\theta} \in \mathcal{C}\}$ defines a PC, and hence induces a probability measure $p(\cdot \mid \boldsymbol{\theta}, \mathcal{G})$. We can then compute bounds on the conditional expectations of a function over some variable X_i

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(f|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}) = \min_{\boldsymbol{\theta}} \sum_{x_i \in \mathcal{X}_i} f(x_i) p(X_i = x_i | \mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}, \boldsymbol{\theta}, \mathcal{G})\,. \tag{7.2}$$

The equation above is well-defined if $\min_{\boldsymbol{\theta}} p(\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}, \mathcal{G}) > 0$, which we will assume to be true[2]. Note also that we can focus on the computation of the lower expectation, as the upper expectation can be obtained from

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(f|\mathbf{x}_{\mathcal{E}}) = -\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(-f|\mathbf{x}_{\mathcal{E}}).$$

Computing the lower conditional expectation in Equation equation 7.2 is facilitated when we have PCs with tree structures. In that case, for some real μ we have the following relation

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}(f|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}) > \mu \iff \min_{\boldsymbol{\theta}} \sum_{x_i \in \mathcal{X}_i} (f(x_i) - \mu) p(X_i = x_i, \mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}, \mathcal{G}) > 0\,.$$

[2]Statistical models often have some smoothing so that zero probability is not attributed to any assignment of variables. That is our assumption here for the sake of simplicity, though this could be addressed in more sophisticated ways.

That allows us to obtain the exact value (to the desired precision) of the minimisation via a binary search for μ until $\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\theta}}((f-\mu)|\mathbf{x}_{\mathcal{E}}) = 0$. In particular, for PCs with tree-shaped structure, there is a polynomial-time algorithm to compute such expectations of univariate functions. The algorithm runs in a single pass through the network, like the minimisation algorithm described above, but has to keep track of both minimum and maximum values because an arbitrary function f is not constrained to be non-negative everywhere like probability functions. We refer the reader to (Maua et al. 2018) for more details.

7.3 Efficient Robustness Measure Computation

We now define an architecture called class-selective (C)PC that is provenly more efficient in computing robustness values.

Definition 6 (Class-selective PC)
Consider a domain where variable Y is called the class variable. A class-selective (C)PC has a sum node as root node with $|\mathcal{Y}|$ product nodes as its children, each of which has an indicator leaf node for each possible value for Y (besides potentially other sibling (C)PCs). These product nodes that are children of the root sum node have disjoint sets of internal descendant nodes.

The name class-selective was inspired by selective SPNs (Peharz et al. 2014), where only one child of each sum node is active (has output larger than zero) at a time. As described in Chapter 5, that property is often called *determinism* (Darwiche 2003; Vergari et al. 2020). A class-selective PC is only deterministic at the root node: for a given class value, only one of the sub-networks remains active. That is depicted in Figure 7.1, where only one indicator node λ_i is non-zero and all but one of the children of the root node evaluate to zero.

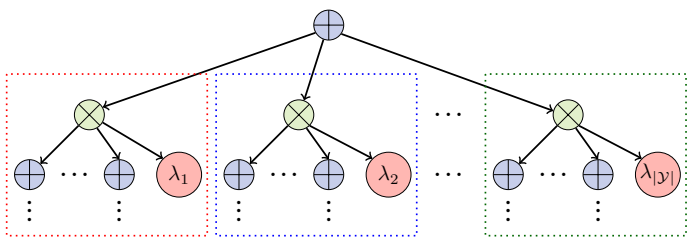

Figure 7.1: Illustration of the first (top) layer of a class-selective PC. In the graph, λ_i is a leaf node applying the indicator function $\mathbb{1}(Y = i)$.

In a class-selective CPC, the computation of the expectation of a function of the class variable can be achieved at the same cost of likelihood evaluation in standard PCs: in a single pass through the network.

$$\min_{\boldsymbol{\theta}_r} \sum_y f(y) p_r(Y = y, \mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}_r) =$$

$$\min_{\boldsymbol{\theta}_r} \left[\sum_{y : f(y) \geq 0} f(y) w_{r,y} \min_{\boldsymbol{\theta}_y} p_y(y, \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}_y) + \sum_{y : f(y) < 0} f(y) w_{r,y} \max_{\boldsymbol{\theta}_y} p_y(y, \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}_y) \right],$$

where r is the root node and, with a slight abuse of notation, $y \in \mathcal{Y}$ indexes the child of the root node associated with class y. Note that each of these internal optimisations can be obtained by independent executions which take altogether time $O(|\boldsymbol{\theta}|)$ by Corollary 5 (as each execution runs over non-overlapping sub-CPCs corresponding to different class labels).

We also note that class-selective PCs offer more efficient classification in the standard, non-credal setting as well. To find the class of maximum probability (and its probability) in a class-selective PC we only have to visit each node once, since we can compute $p(y \mid \mathbf{x}_{\mathcal{E}})$ for every $y \in \mathcal{Y}$ in a single pass through the network. Conversely, in a non-class-selective PC we would have to visit each node $|\mathcal{Y}|$ times because we need one pass for each probability $p(y \mid \mathbf{x}_{\mathcal{E}})$. Of course, this does not account for how the two architectures might differ in number of parameters; non-class-selective PCs might be more compact and thus compensate for the slightly higher computational complexity of performing classification. We empirically evaluate and discuss some of these aspects in Section 7.5 and Table 7.2.

Let us now turn our attention to the CPC robustness estimation in a classification problem. Given input instance $\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}$ for which we want to predict the class variable, we say that a classification issued by CPC C is robust if the class value $y = \arg\max_{\mathcal{Y}} p(y|\mathbf{x}, \boldsymbol{\theta}, \mathcal{G})$ predicted by a PC that belongs to a certain CPC $C = \{p(\cdot \mid \boldsymbol{\theta}, \mathcal{G}) : \boldsymbol{\theta} \in \mathcal{C}\}$ is also the prediction of any other $p(\cdot \mid \boldsymbol{\theta}', \mathcal{G}) \in C$ (hence it is unique for C). This is often called *credal dominance* or *credal classification* (Zaffalon 2002), and can de defined as

$$\min_{\boldsymbol{\theta}} [p(y|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}) - p(y'|\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})] > 0 \text{ for every } y' \neq y\,. \tag{7.3}$$

In the case of e-contaminated class-selective CPCs, it suffices to check whether

$$\min_{\boldsymbol{\theta}} p(Y = y, \mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta}) > \max_{y' \in \mathcal{Y} : y' \neq y} \max_{\boldsymbol{\theta}} p(Y = y', \mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})\,,$$

that is, regardless of the choice of weights $\boldsymbol{\theta} \in \mathcal{C}$, we would have a stable prediction y with $p(y|\mathbf{x}, \boldsymbol{\theta}) > p(y'|\mathbf{x}, \boldsymbol{\theta})$ for all other labels y'.

General CPCs may require $2 \cdot |\,\boldsymbol{\theta}\,| \cdot (|\mathcal{Y}| - 1)$ node evaluations in the worst case to identify whether a predicted class label y is robust for instance $\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}$, while a class-selective CPC obtains such result in $|\,\boldsymbol{\theta}\,|$ node evaluations. This is because CPCs will run over its nodes $(|\mathcal{Y}| - 1)$ times in order to reach a conclusion about the expression in Equation 7.3, while the class-selective CPC can compute $\min_{\boldsymbol{\theta}} p(\mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})$ and $\max_{\boldsymbol{\theta}} p(\mathbf{x}_{\mathcal{E}} \mid \boldsymbol{\theta})$ (the $\max$ is done for each y') in a single pass, taking overall $|\,\boldsymbol{\theta}\,|$ node evaluations, since they run over non-overlapping sub-networks for different class values.

Finally, given an input instance $\mathbf{X}_{\mathcal{E}} = \mathbf{x}_{\mathcal{E}}$ and a PC $p(\cdot \mid \boldsymbol{\theta}, \mathcal{G})$ learnt from data, we can compute a robustness measure as follows. We define a collection of CPCs $C_{\boldsymbol{\theta},\epsilon}$ parametrised by $0 \leq \epsilon < 1$ such that each $\boldsymbol{w}_v^{\epsilon}$ of a node v in $C_{\boldsymbol{\theta},\epsilon}$ is allowed to vary within an ϵ-contaminated credal set of the original weight vector $\boldsymbol{w}_v \in \boldsymbol{\theta}$ of the same node v. We use the same notation for the collection of parameters $\boldsymbol{\theta}$ of a PC, and denote $\boldsymbol{\theta}_v^{\epsilon}$ the ϵ-contaminated credal set of $\boldsymbol{\theta}_v$. A robustness measure for a prediction issued by the original PC $\hat{y} = \arg\max_{\mathcal{Y}} p(y \,|\, \mathbf{x}_{\mathcal{E}}\, \boldsymbol{\theta}, \mathcal{G})$ is then defined by the largest ϵ such that $C_{\boldsymbol{\theta},\epsilon}$ is robust for $\mathbf{x}_{\mathcal{E}}$, that is, the largest ϵ for which every PC in $C_{\boldsymbol{\theta},\epsilon}$ predicts the same class label $\hat{y}$. Finding such ϵ can be done using a simple binary search, as shown in Algorithm 4.

7.4 Likelihood as a Measure of Reliability

PCs also facilitate exact computation of the (log-)likelihood of the explanatory variables $\mathbf{X}$, which is a useful tool to identify outliers. In a classification context, for a given instance $\mathbf{x}$, the lower its likelihood under the model $p(\mathbf{x} \mid \boldsymbol{\theta}, \mathcal{G})$, the less we should trust the prediction $\hat{y} = \arg\max_{\mathcal{Y}} p(y \,|\, \mathbf{x}, \boldsymbol{\theta}, \mathcal{G})$. That is because a low likelihood indicates $\mathbf{x}$ is unlikely to have been sampled from the same distribution as the training instances.

Unfortunately, this simple and reasonable notion of reliability is known to be problematic when the likelihood is computed by deep generative models (Nalisnick et al. 2018), even tractable ones like normalising flows (Papamakarios et al. 2021). Yet, the likelihood computed by PCs seem to be a strong signal to detect outliers as demonstrated in (Peharz et al. 2020b) and in our own work that we discuss in this chapter. Moreover, monitoring the marginal $p(\mathbf{x}_{\mathcal{E}})$ for some arbitrary partial evidence $\mathbf{x}_{\mathcal{E}}$ in PCs is rather efficient. In the case of class-selective PCs, this comes at no extra cost; these models are designed to compute $p(y, \mathbf{x}_{\mathcal{E}})$

```
Function Robustness(p(· | θ, G), x_E, y, er):
    Data   : Class-selective PC p(· | θ, G), Input x, Prediction
             y | X_E = x_E, Precision er < 1
    Result: Robustness ϵ
    ϵ_max ← 1;
    ϵ_min ← 0;
    while ϵ_min < ϵ_max − er do
        ϵ ← (ϵ_min + ϵ_max)/2;
        v ← min_{θ^ϵ} p(y, x_E | θ^ϵ);
        for y' ≠ y do
            v' ← max_{θ^ϵ} p(y', x_E | θ^ϵ);
            if v' ≥ v then
                ϵ_max ← ϵ;
                break
            end
        end
        if ϵ_max > ϵ then
            ϵ_min ← ϵ
        end
    end
    return ϵ;
```

Algorithm 4: Efficient ϵ-robustness computation.

for each $y \in \mathcal{Y}$ in a single pass through the network, and thus computing $p(\mathbf{x}_\mathcal{E})$ only requires summing these individual probabilities, $p(\mathbf{x}_\mathcal{E}) = \sum_y p(y, \mathbf{x}_\mathcal{E})$.

We can also efficiently compute $p(\mathbf{x}_\mathcal{E})$ with GeFs, but we need a slight adaptation. As defined in Equation 6.3, GeFs are an ensemble of generative GeDTs and thus do not encode a single full joint distribution. However, we can extend GeFs to model a single joint by considering a uniform mixture of GeDTs (using a sum node at the root with all GeDTs as children), instead of an ensemble of the conditional distributions of each GeDT. In this case, the model represents the joint

$$p(\mathbf{x}, y) = \sum_{v \in \mathrm{ch}(r)} w_{r,v} p_v(\mathbf{x}, y) = \frac{1}{n_t} \sum_{v \in \mathrm{ch}(r)} p_v(\mathbf{x}, y),$$

where n_t is the number of GeDTs in the ensemble, r is the root node with uniform weights $w_{r,v} = 1/n_t$, and each p_v is defined by a different GeDT. This model, to which we shall refer as GeF$^+$, allows us to detect outliers and achieves

similar but slightly inferior performance than GeFs in classification with missing data (still clearly superior to KNN imputation). This does not come as a surprise: the benefits of a fully generative model often comes at the cost of a (small) drop in classification accuracy.

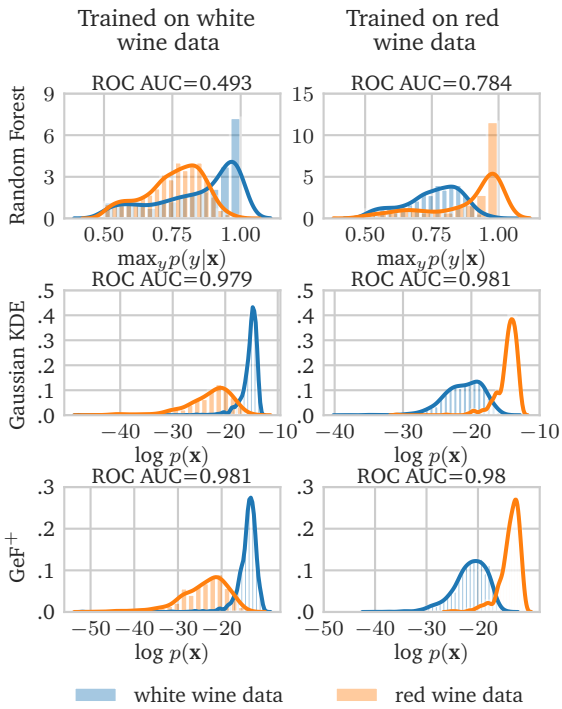

Figure 7.2: Comparison of different outlier detection methods. Normalised histograms of $\log p(\mathbf{x})$ (KDE and GeF$^+$) and $\max_y p(y|\mathbf{x})$ (Random Forest) of samples from red and white wine data.

We illustrate outlier detection with GeF$^+$s using the wine quality dataset (Cortez et al. 2009) (where the class is a scale of quality of wine) with a variant of transfer testing (Bradshaw et al. 2017). We learn two different GeF$^+$s, each with only one type of wine (red or white), and compute the log-density of unseen data (70/30 train-test split) for the two wine types. As we see in the

histograms of Figure 7.2, the marginal distribution over the joints does provide a strong signal to identify out-of-domain instances. We compare GeF$^+$s to a Gaussian Kernel Density Estimator (KDE) and to a common baseline for deep models (Hendrycks and Gimpel 2016), whereby the probability of the predicted class, $\max_y p(y|\mathbf{x})$, is used as a signal to detect outliers. We see from the histograms and the ROC (receiver operating characteristic curve) scores, that our models largely outperform the baseline while being comparable to KDEs, even though the structure of a GeF$^+$ is learnt in a discriminative manner.

We note that previous works have already proposed using Random Forests for outlier detection (Liu et al. 2008), but these models are directly trained to identify anomalies and have that as their sole purpose. GeF$^+$s are unique in that, while being primarily classifiers, they also effectively detect outliers.

7.5 Experiments

We investigated the performance of class-selective (C)PCs and GeF$^+$s through a series of experiments over a range of 19 UCI datasets (Dua and Graff 2017). We also use the same datasets to study how ϵ-robustness relates to accuracy as well as log-likelihood. All experiments were run in a single modern core with our implementation of GeFs and Credal PCs, the latter of which runs LearnSPN (Gens and Domingos 2013) for structure learning. The source code for class-selective (C)PCs is available at `https://github.com/alcorreia/sum2019`, whereas the source code for GeF$^+$s is hosted at `https://github.com/alcorreia/gefs`.

Table 7.1 presents the UCI datasets on which we ran experiments described by their number of independent instances n, number of variables m (including a class variable Y) and number of class labels $|\mathcal{Y}|$. All datasets are categorical (or have been made categorical using discretisation by median value). We also show the classification accuracy obtained by both class-selective and non-class-selective (denoted general) PCs as well as GeF$^+$s and XGBoost (Chen and Guestrin 2016), considered a state-of-the-art technique for supervised classification tasks. We used the XGBoost implementation from scikit-learn (Pedregosa et al. 2011) with default hyperparameters and 100 trees (estimators). For GeF$^+$s we used 100 trees and fully factorised leaves, while keeping other hyperparameters as in (Correia et al. 2020b). Results across all datasets are obtained by stratified 5-fold cross-validation.

As one can inspect, class-selective PCs outperformed general PCs in 14 out of these 19 datasets, while being comparable to XGBoost and GeF$^+$s. This shows the class-selective architecture does introduce an effective inductive bias that

Table 7.1: Percent accuracy of XGBoost, General PCs and Class-selective PCs across several UCI datasets. Both PC types were learnt via LearnSPN (Gens and Domingos 2013). All experiments consisted in stratified 5-fold cross validation.

Dataset	n	m	$\lvert\mathcal{Y}\rvert$	XGBoost	General PC	Class-selective PC	GeF$^+$
bridges	107	11	6	64.486	57.009	63.551	65.421
autos	205	26	2	90.244	88.293	89.268	89.268
breast cancer	286	10	2	68.182	71.678	66.783	70.629
heart h	294	12	2	78.912	79.932	81.973	79.932
liver disorders	345	7	2	67.246	57.391	62.609	68.116
dermatology	366	35	6	96.721	81.694	98.907	98.634
colic	368	23	2	82.609	77.717	73.37	83.967
balance scale	625	5	3	71.36	72.48	72	73.28
soybean	683	36	19	92.826	62.518	93.704	94.729
diabetes	768	9	2	72.005	70.703	70.964	70.833
vehicle	846	19	4	66.312	46.454	65.248	64.657
tic tac toe	958	10	2	93.946	69.937	89.562	98.121
vowel	990	14	11	72.02	33.737	61.919	75.152
solar flare 2	1,066	12	6	74.203	59.475	75.328	74.015
cmc	1,473	10	3	52.003	48.133	47.115	49.219
car	1,728	7	4	98.843	70.023	94.502	97.222
segment	2,310	17	7	83.42	67.662	82.641	82.987
sick	3,772	28	2	93.558	93.876	92.497	93.505
spambase	4,601	8	2	79.961	78.505	79.961	79.135

favours classification. Of course, this benefit is limited to a single a variable, since we design the entire structure around the target class, but discriminative models, like XGBoost, are equally oriented towards a single variable. Yet, class-selective PCs as well as GeF$^+$s are fully generative models capable of, among other things, outlier detection and handling missing data besides being good classifiers. In particular, GeF$^+$s outperformed XGBoost in 10 of the 19 datasets.

We further contrast general and class-selective networks in Table 7.2, where we compare the two types of network in terms of their architecture and processing times on classification tasks. One can see that class-selective PCs have a higher number of parameters due to a larger number of sum nodes. However, in some cases general PCs are deeper, which means class-selective PCs tend to grow sideways, especially when the number of classes is high. Nonetheless, the larger number of parameters in class-selective networks does not translate into higher latency as both architectures have similar learning and inference times. We attribute that to the independence of the subnetwork of each class which facilitates inference. Notice that the two architectures are equally efficient only in the classification task (only aspect compared in Table 7.2) and not on robustness computations. We mathematically proved the latter to be more efficient in class-selective networks when using Algorithm 4.

Table 7.2: Comparison between General (Gen) and Class-Selective (CS) PCs in learning and average inference times (s), height, and number of nodes and parameters.

Dataset	N	$\|X\|$	$\|X_c\|$	Learning (s)		Inference (s)		# Nodes		Height		# Params.	
				Gen	CS	Gen	CS	Gen	CS	Gen	CS	Gen	CS
bridges	107	11	6	0.228	0.358	$1.529e{-2}$	$2.74e{-2}$	154	289	7	5	39	71
autos	205	26	2	1.644	1.652	$2.953e{-2}$	$3.02e{-2}$	958	971	12	11	211	221
breast cancer	286	10	2	0.382	0.418	$7.835e{-3}$	$9.793e{-3}$	253	304	9	9	51	59
heart h	294	12	2	0.22	0.21	$4.193e{-3}$	$4.307e{-3}$	131	138	6	6	37	39
liver disorders	345	7	2	0.107	0.108	$2.467e{-3}$	$2.5e{-3}$	77	78	6	6	24	25
dermatology	366	35	6	2.971	2.802	0.161	0.171	1,834	1,952	15	10	383	408
colic	368	23	2	1.084	1.326	$1.885e{-2}$	$2.405e{-2}$	625	791	11	12	149	183
balance scale	625	5	3	0.11	0.112	$4.258e{-3}$	$4.068e{-3}$	85	82	7	6	26	24
soybean	683	36	19	4.308	4.969	0.763	1.125	2,604	3,913	16	9	596	940
diabetes	768	9	2	0.263	0.252	$5.672e{-3}$	$5.564e{-3}$	176	172	9	8	58	56
vehicle	846	19	4	1.075	1.482	$3.717e{-2}$	$5.226e{-2}$	585	830	12	11	186	272
tic tac toe	958	10	2	0.725	0.66	$1.623e{-2}$	$1.568e{-2}$	496	470	11	11	128	116
vowel	990	14	11	3.498	3.879	0.417	0.49	2,444	2,800	17	13	502	663
solar flare 2	1,066	12	6	1.03	0.827	$6.738e{-2}$	$6.088e{-2}$	784	708	12	10	158	148
cmc	1,473	10	3	1.156	1.136	$3.775e{-2}$	$3.685e{-2}$	828	812	15	14	198	193
car	1,728	7	4	0.523	0.556	$2.337e{-2}$	$2.794e{-2}$	370	434	11	10	81	92
segment	2,310	17	7	1.558	2.028	$9.55e{-2}$	0.142	888	1,301	13	11	263	419
sick	3,772	28	2	2.665	2.229	$2.493e{-2}$	$1.926e{-2}$	802	616	13	11	249	196
spambase	4,601	8	2	0.649	0.599	$1.205e{-2}$	$1.247e{-2}$	353	364	10	12	115	119

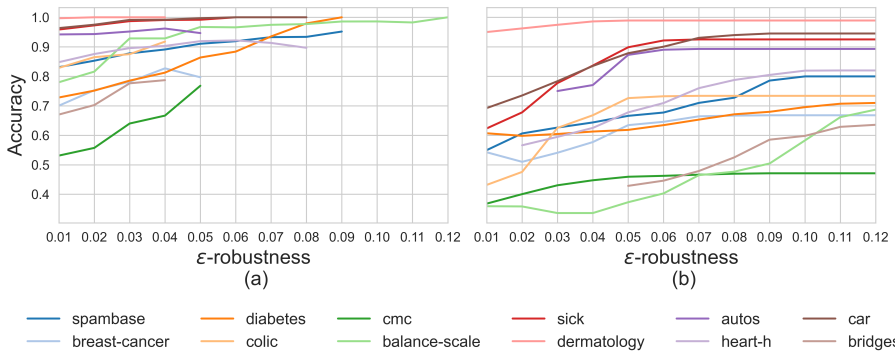

Figure 7.3: Accuracy of class-selective Probabilistic Circuits on predictions with robustness (a) *above* and (b) *below* different thresholds for 12 UCI datasets (Dua and Graff 2017). Some curves end abruptly because we only computed the accuracy when 50 or more data points were available for a given threshold.

7.5.1 Exploring Data on the Robustness Measure

We can interpret robustness as a measure of how confident a model is on a given prediction. Roughly speaking, in a classification task, the robustness value ϵ of a prediction corresponds to how much we can tweak the networks parameters without changing the final result, i.e., the class of maximum probability. Thus, a large ϵ means that many similar networks (in parameter space) would give the same output for the instance in question. Similarly, we can think that small changes in the training data or in the hyperparameters of the learning algorithm would not produce a model whose prediction would be different for that given instance. Conversely, a small ϵ tell us that slightly different networks would provide us with a distinct answer. In that case, the prediction is not reliable as it might fluctuate with any variation on the learning or data acquisition processes.

We can validate this interpretation by investigating how robustness relates to the accuracy of the model. In Figure 7.3(a), we defined a number of robustness thresholds and, for each of them, we computed the accuracy of a class-selective PC over instances for which ϵ was *above* the threshold. It is clear from the graph that the accuracy increases with the threshold, and we can infer that robustness does translate into reliability since the model is more accurate over instances with high ϵ values. We arrive at a similar conclusion in Figure 7.3(b), where we plot instances for which ϵ was *below* a given threshold. In this case, the curves

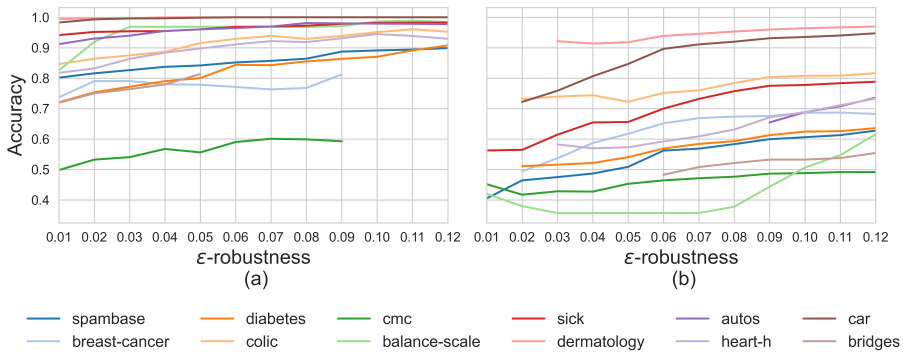

Figure 7.4: Accuracy of GeF$^+$s on predictions with robustness (a) *above* and (b) *below* different thresholds for 12 UCI datasets (Dua and Graff 2017). Some curves end abruptly because we only computed the accuracy when 50 or more data points were available for a given threshold.

start at much lower accuracy values, with only examples of low robustness, and then build up as instances with higher ϵ values are added. Finally we repeated the same experiment with GeF$^+$s and, as shown in Figure 7.4, we observed the same pattern with ϵ-robustness closely related to accuracy.

7.6 Contrasting Robustness and Likelihood

We also trained GeF$^+$s on the Mnist (LeCun et al. 2010) and Fashion-Mnist (Xiao et al. 2017) datasets to visually inspect samples with different ϵ-robustness values. In Figure 7.5, we report test instances with lowest and highest ϵ-robustness for each dataset, with correctly classified images in green and incorrectly classified images in red. We see that samples with low robustness values are not only less likely to be correctly classified but also often contain irregular shapes and patterns. These results support the idea that ϵ-robustness is a decent measure of uncertainty, since it seems to identify instances that are hard to classify.

We emphasise outlier detection and robustness estimation are related but different notions, and GeF$^+$effectively distinguishes them. Figure 7.6 shows a few of the most likely and unlikely (Fashion-)Mnist samples under the training data distribution. While samples are ordered by their marginal density $p(\mathbf{x})$, the background light is proportional to their ϵ-robustness, with darker colours for

larger ϵ values. We can clearly see how these measures differ as, for example, although the model deems 1s highly likely, ϵ-robustness seems to vary with the shape/orientation of the trace.

Moreover, these two measures complement each other and allow us to better understand the underlying cause of the model's uncertainty. Notably, for a consistent model—one that fits the true data-generating distribution if given sufficient data—and a sample $\mathbf{x}$ with high $p(\mathbf{x})$ and low ϵ-robustness, one may infer there is high aleatoric uncertainty. A number nine with an incomplete circle at the top is a good example of a pattern in handwritten digits that, albeit likely, is still hard to tell apart from a number four. Conversely, an instance might

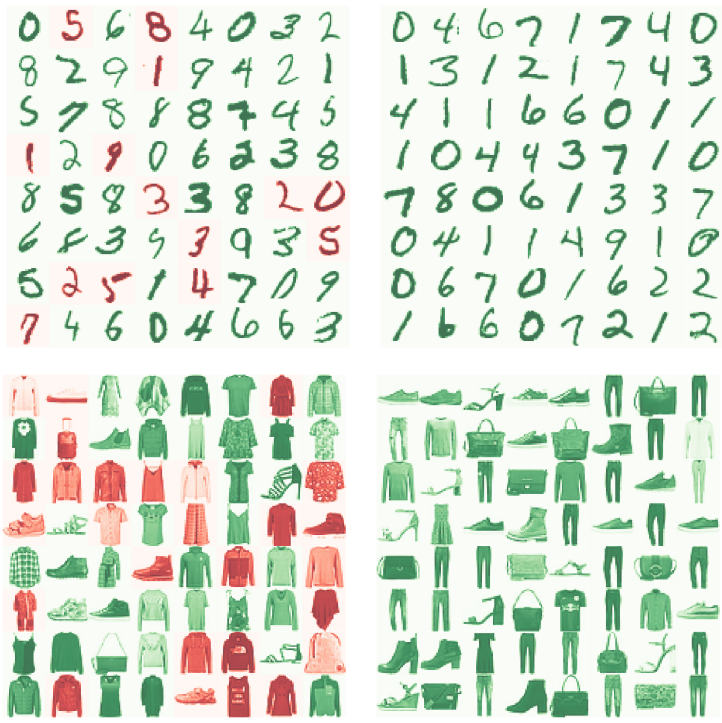

Figure 7.5: Test samples from (Fashion-)Mnist datasets with lowest (left) and highest (right) ϵ-robustness computed by a trained GeF$^+$ model. Correctly and incorrectly classified examples are shown in green and red, respectively.

be misshaped and hence unlikely, but still be associated with high robustness values. In that case, epistemic uncertainty is dominant, that is, the model has not been trained on similar examples and its high confidence estimate should not be trusted. Distinguishing the two types of uncertainty is not only insightful to better understand the behaviour of the model but also helps establish the correct course of action. Namely, we are better off suspending judgement when faced with aleatoric uncertainty, whereas cases of epistemic uncertainty can be addressed by collecting more data and possibly retraining the model.

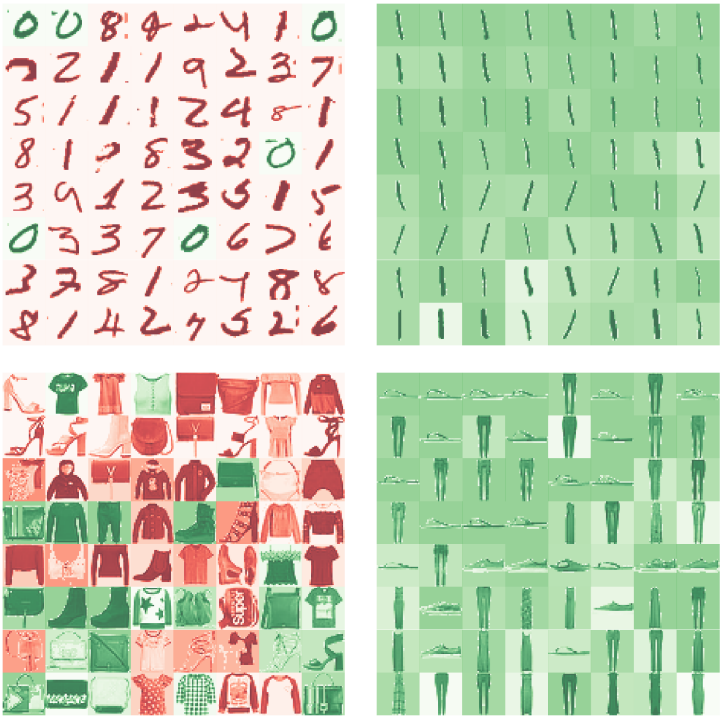

Figure 7.6: Test samples from (Fashion-)Mnist datasets with lowest (left) and highest (right) $p(\mathbf{x})$ in a trained GeF$^+$ model. The background light is proportional to the ϵ-robustness of each instance.

7.7 Conclusion

Probabilistic Circuits and their credal counterparts are expressive deep generative models with tractable inference, which makes them a promising option for large-scale applications. In this chapter, we have proposed a new architecture, class-selective (C)PCs, that combine efficient robustness computation with high accuracy on classification tasks, outperforming general (C)PCs. Even though they excel in discriminative tasks, class-selective PCs as well as GeFs are still generative models fully endowed with the semantics of graphical models. We demonstrated how their probabilistic semantics can be brought to bear through their extension to Credal PCs, which allows us to define and compute a robustness measure named ϵ-robustness. Empirically, we explored how ϵ-robustness relates to the accuracy and log-likelihood of the model, showing it is a promising alternative to quantify the reliability of each prediction.

We finally point out some interesting directions for future work. As demonstrated here, class-selective (C)PCs have proven to be powerful models in classification tasks, but they arbitrarily place the class variable in a privileged position in the network. Future research might investigate how well class-selective (C)PCs fit the joint distribution and how they would fair in predicting other variables. Another promising avenue for future research is to explore using different ϵ values for different parts of the architecture. Using a single value might not yield the best uncertainty measure since perturbing nodes closer to the root is likely to produce larger variations in the prediction, and thus much of the robustness analysis might be dominated by these nodes on the top.

Chapter 8

Continuous Mixtures of Tractable Probabilistic Models

Probabilistic models based on continuous latent spaces, such as variational autoencoders, can be understood as uncountable mixture models where components depend continuously on the latent code. They have proven to be expressive tools for generative and probabilistic modelling, but are at odds with tractable probabilistic inference, that is, computing marginals and conditionals of the represented probability distribution. Meanwhile, tractable probabilistic models such as probabilistic circuits (PCs) can be understood as hierarchical discrete mixture models, and thus are capable of performing exact inference efficiently but often show subpar performance in comparison to continuous latent-space models. In this chapter, we investigate a hybrid approach, namely continuous mixtures of tractable models with a small latent dimension. While these models are analytically intractable, they are well amenable to numerical integration schemes based on a finite set of integration points. With a large enough number of integration points the approximation becomes de-facto exact. Moreover, for a finite set of integration points, the integration method effectively compiles the continuous mixture into a standard PC. In experiments, we show that this simple scheme proves remarkably effective, as PCs learnt this way set new state of the art for tractable models on many standard density estimation benchmarks.

This chapter is almost integrally based on **Alvaro Correia**, Gennaro Gala, Erik Quaeghebeur, Cassio de Campos and Robert Peharz: Continuous Mixtures of Tractable Probabilistic Models, *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 37, No. 01, 2023.

8.1 Introduction

Probabilistic modelling typically aims to capture the data-generating joint distribution, which can then be used to perform probabilistic inference to answer queries of interest. A recurring scheme in probabilistic modelling is the use of an *uncountable mixture model*, that is, the data generating distribution is approximated by

$$p(\mathbf{x}) = \mathbb{E}_{p(\mathbf{z})}\left[p(\mathbf{x} \mid \mathbf{z})\right] = \int p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z} \tag{8.1}$$

where $p(\mathbf{x})$ is the modelled density over variables $\mathbf{X}$, $p(\mathbf{z})$ is a mixing distribution (prior) over latent variables $\mathbf{Z}$, and $p(\mathbf{x} \mid \mathbf{z})$ is a conditional distribution of $\mathbf{x}$ given $\mathbf{z}$, also called a mixture component.

Some successful recent examples of uncountable mixtures are variational autoencoders (VAEs) (Kingma and Welling 2014), generative adversarial networks (GANs) (Goodfellow et al. 2014), and normalising Flows (Rezende and Mohamed 2015). All three of these models use a simple prior $p(\mathbf{z})$, e.g. an isotropic Gaussian, and represent the mixture components with a neural network. In the case of VAEs, the mixture component is a bona-fide distribution $p(\mathbf{x} \mid \mathbf{z})$ represented by the the so-called decoder, while for GANs and Flows the mixture component is a point measure, i.e. a deterministic function $\mathbf{x} = f(\mathbf{z})$[1]. The use of continuous neural networks topologically relates the latent space and the observable space with each other, so that these models can be described as *continuous* mixture models. The use of continuous mixtures allows to a certain extent the interpretation of $\mathbf{Z}$ as a (latent) embedding of $\mathbf{X}$ but also seems to be beneficial for generalisation, i.e. to faithfully approximate real-world distributions with rather little training data.

However, while continuous mixture models have shown impressive results in density estimation and generative modelling, their ability to perform probabilistic inference remains rather limited. In particular, the key inference routines of *marginalisation and conditioning*, which together form a consistent reasoning process (Ghahramani 2015; Jaynes 2003), are generally intractable in these models, mainly due to the integral in Equation 8.1 which forms a hard computational problem in general.

Meanwhile, the area of *tractable probabilistic modelling*, aims for models which allow for a wide range of exact and efficient inference routines. In that

[1] In normalising Flows the vectors $\mathbf{z}$ and $\mathbf{x}$ depend on each other via a bijection, so $\mathbf{z}$ is not exactly latent.

arena, Probabilistic Circuits (or PCs) are probably the most prominent framework, encompassing models that support efficient marginalisation and conditioning operations. As discussed in Chapter 5, PCs are computational graphs defined by a structure $\mathcal{G}$ and a set of parameters $\boldsymbol{\theta}$, comprising the weights of its sums nodes and the parameters of its distribution nodes. In this chapter, we are particularly interested in two specific PC structures corresponding to factorised distributions and Chow-Liu trees. While these structures are arguably simplistic, we will show in the experiments section that continuous mixtures of such PCs outperform all state-of-the-art PC learners on 16 out of 20 common benchmark datasets.

From a representational point of view, PCs can be interpreted as hierarchical, discrete mixture models (Peharz 2015; Peharz et al. 2016; Zhao et al. 2016), i.e. they compute a distribution function that can be generally written as

$$p(\mathbf{x}) = \sum_{\mathbf{z}'} p(\mathbf{x} \mid \mathbf{z}')\, p(\mathbf{z}') \tag{8.2}$$

where $\mathbf{Z}'$ is a *discrete* (latent) vector, but otherwise the form is similar to the continuous mixture in Equation 8.1. The number of states of $\mathbf{Z}'$ and thus the number of represented mixture components $p(\mathbf{x} \mid \mathbf{z}')$ grows exponentially in the depth of the PC (Peharz 2015; Zhao et al. 2016). Moreover, recent vectorisation-based implementations (Peharz et al. 2020a) have enabled large scale PCs (>100M of parameters) at execution speeds comparable to standard neural networks. These endeavours evidently boosted the performance of tractable models, but there is still a significant gap to intractable models such as VAEs. One reason for this performance gap is likely the structural constraints in PCs, which are required to maintain tractability but are at odds with expressivity. This tension between tractability and expressiveness is a common folklore in the tractable inference community.

On the other hand, a *huge* discrete mixture model of the form of Equation 8.2 should in principle be able to outperform moderately sized uncountable mixtures. Yet, on vanilla benchmarks, we do not see this result. For instance, a vanilla VAE with a few million parameters easily gets test log-likelihoods higher than -90 nats on Binary MNIST (Tomczak and Welling 2018), whereas an Einet with 84 million parameters (Peharz et al. 2020a) barely gets above -100 nats (or equivalently 0.184 bits per dimension), as shown in Table 8.6. Thus, a complementary explanation is that discrete (and hierarchical) mixtures—like PCs—are hard to learn (or generalise poorly), while continuous mixtures—like VAEs—are easier to learn (or generalise well).

In this chapter, we follow a hybrid approach and consider *continuous mixtures of tractable models*. In particular, we consider continuous mixtures of two very simple tractable models, namely *completely factorised distributions* ($\mathcal{G}_{\mathsf{F}}$ structure) and *Chow-Liu Trees* ($\mathcal{G}_{\mathsf{CLT}}$ structure), both of which can be easily expressed as PCs. Specifically, we consider models of the form

$$p(\mathbf{x}) = \mathbb{E}_{p(\mathbf{z})}\left[p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))\right] \tag{8.3}$$

where $p(\mathbf{z})$ is an isotropic Gaussian prior and $p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))$ is a PC with its parameters depending on $\mathbf{z}$ via some neural-network $\boldsymbol{\theta}(\mathbf{z})$. While these models are analytically intractable, we can approximate the marginalisation of $\mathbf{z}$ arbitrarily well with numerical techniques, such as *Gaussian quadrature rules* and *(quasi) Monte Carlo*. The common principle of these methods is that they select a finite set of *integration points* $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ in either a deterministic or (partially) random manner and construct a corresponding weight function $w(\mathbf{z})$ such that Equation 8.3 is approximated as

$$p(\mathbf{x}) = \mathbb{E}_{p(\mathbf{z})}\left[p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))\right] \approx \sum_{i=1}^{N} w(\mathbf{z}_i)\, p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i)). \tag{8.4}$$

All integration methods we consider become exact for $N \to \infty$ and under certain conditions on $\boldsymbol{\theta}(\mathbf{z})$ one can derive guarantees for the approximation error for $N < \infty$. In particular, for (quasi) Monte Carlo integration it is straightforward to derive probabilistic error guarantees for the approximation quality by leveraging concentration bounds. Moreover, an empirical observation is that numerical integration works reasonably well for low dimensional spaces, but tends to deteriorate for larger dimensionality. Thus, in the experiments that follow we keep the dimensionality of $\mathbf{Z}$ relatively small (≤ 16), so that our continuous mixtures of tractable models remain 'morally tractable'.

Of practical relevance is that, for the numerical methods considered in this work, the integration weights $\{w(\mathbf{z}_i)\}_{i=1}^{N}$ sum to one[2], so that the approximation in Equation 8.4 can be interpreted as a discrete mixture model. Additionally, for fixed $\mathbf{z}_i$, each $p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i))$ is simply a PC with fixed parameters $\boldsymbol{\theta}_i = \boldsymbol{\theta}(\mathbf{z}_i)$, so that Equation 8.4 in fact yields a mixture of PCs, which in turn can be again interpreted as a large PC. Thus, we can convert a learnt intractable model from Equation 8.3 into an approximate PC which facilitates exact inference, that is, 'performing exact inference in an approximate model'. This principle is illustrated in Fig. 8.1.

[2]For Monte Carlo, the weights are naturally given by $w(\mathbf{z}_i) = 1/N$.

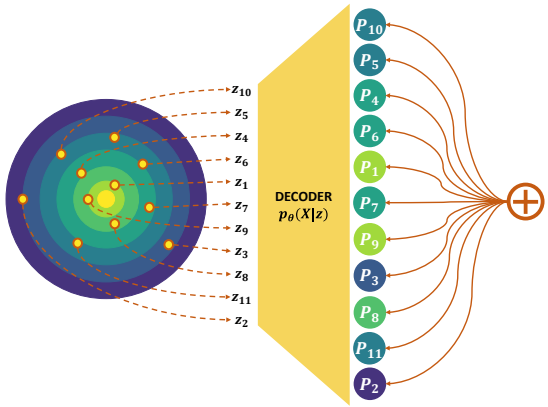

Figure 8.1: Continuous mixture model evaluated with $N{=}11$ integration points $\{\mathbf{z}_i\}_{i=1}^N$. Each P_i is a tractable probabilistic model with parameters depending on $\mathbf{z}_i$.

To the best of our knowledge, this simple idea has not been explored before, and yet it delivers astonishing results: On 16 out of 20 of commonly used density estimation benchmarks, we set new state of the art in the domain of tractable models, outperforming even the most sophisticated PC structure and parameter learners. We also achieve competitive results on image datasets, where in comparison to other PCs, our models produced better samples and often attained better test log-likelihoods.

8.2 Related Work

The perhaps most widely known *continuous mixture model* are *variational autoencoders* (VAEs) (Kingma and Welling 2014; Rezende and Mohamed 2015), specifying the model density as $p(\mathbf{x}) = \int p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))\, p(\mathbf{z}) \mathrm{d}\mathbf{z}$ where $p(\mathbf{z})$ is an isotropic Gaussian and $p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))$ is typically a fully factorised distribution of Gaussians or Binomials with parameters provided by a neural network $\boldsymbol{\theta}(\mathbf{z})$, the so-called decoder, taking $\mathbf{z}$ as input. Since the latent code in VAEs is usually relatively high-dimensional, learning and inference is implemented via learnt amortised inference (Kingma and Welling 2014).

When using a fully factorised structure ($\mathcal{G}_\mathsf{F}$), our models specify in fact the same model as VAEs, which has originally been introduced by McKay (MacKay 1995) under the name *density network*. Our work essentially re-visits McKay's

work, who already mentions: *For a hidden vector of sufficiently small dimensionality, a simple Monte Carlo approach to the evaluation of these integrals can be effective.* In the experiments, we compare numerical integration methods and find that randomised quasi Monte Carlo performs best. Moreover, we also use numerical approximation as a *'compilation approach'*, whereby the continuous mixture is converted into a tractable discrete mixture that sets new state of the art among tractable models in a number of datasets (see Table 8.1).

Similar approaches to our method are HyperSPNs (Shih et al. 2021) and conditional SPNs (Shao et al. 2020), which both use neural nets to compute the weights of PCs. However, HyperSPNs are primarily a regularisation technique that applies to a single PC, whereas in this work we use neural nets to learn continuous mixtures of PCs. In conditional SPNs the parameters are a function of observed variables, while we use a continuous *latent* space in this work.

8.3 Inference and Learning

Our model as specified in (8.3) consists of a continuous latent space $\mathbf{Z}$ and a given PC structure $\mathcal{G}$, whose parameters $\boldsymbol{\theta}(\mathbf{z})$ are a differentiable function of the latent variables. We will broadly refer to function $\boldsymbol{\theta}$ as *decoder* and to the model as a whole as *continuous mixtures*. We use $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ to denote continuous mixtures with factorised and CLT structures, respectively.

Amortised inference (Kingma and Welling 2014; Rezende and Mohamed 2015) is the de-facto standard way to learn continuous mixture models. In this approach, a separate neural network—the so-called *encoder*—represents an approximate posterior $q(\mathbf{z} \mid \mathbf{x})$. The encoder and decoder are learnt simultaneously by maximising the *evidence lower bound* (ELBO)

$$\mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z}) - \log q(\mathbf{z} \mid \mathbf{x}) + \log p(\mathbf{z})], \tag{8.5}$$

which is a lower bound of the (marginal) log-likelihood $\log p(\mathbf{x})$ and thus a principled objective. At the same time, maximising the ELBO is moving q closer to the true posterior in Kullback-Leibler sense, hence tightening the ELBO.

In this work, we investigate numerical integration as an alternative inference and learning method. In particular, we do not require an encoder or any other parametric form of approximate posterior.

8.3.1 Inference via Numerical Integration

Given a function f, a numerical integration method consists of a set of N integration points $\{\mathbf{z}_i\}_{i=1}^N$ and a weight function w such that the integration error $\varepsilon = \left| \int f(\mathbf{z})\,\mathrm{d}\mathbf{z} - \sum_{i=1}^N w(z_i)\, f(z_i) \right|$ is as small as possible. In this chapter, we are interested in approximating the density $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$ of a cm model, so that the integration error with integration points $\{\mathbf{z}_i\}_{i=1}^N$ is given as

$$\varepsilon_{\mathrm{cm}}(\mathbf{x}, \{\mathbf{z}_i\}_{i=1}^N) = \left| \int p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))\, p(\mathbf{z})\, d\mathbf{z} - \sum_{i=1}^N w(\mathbf{z}_i)\, p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i)) \right|.$$

Quadrature Rules

Quadrature Rules divide the integration domain into sub-intervals and approximate the integrand on these intervals with polynomials, which are easy to integrate. They yield a set of deterministic integration points and weights as a function of the degree of the interpolating polynomial. Common quadrature rules like trapezoidal and Simpson's rule achieve error bounds of $\mathcal{O}(N^{-2})$ and $\mathcal{O}(N^{-4})$, respectively. *Gaussian quadrature rules* go a step further and allow us to take into account the distribution of $\mathbf{Z}$; e.g. Gauss-Hermite quadrature is designed for indefinite integrals of the form (8.1) with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Gaussian quadratures integrate exactly any polynomial of degree $2N-1$ or less. That makes them an attractive and general integration technique, since by the Weierstrass approximation theorem (Weierstrass 1885), any function can be approximated by a polynomial to an arbitrary degree of precision, under only mild regularity conditions.

Unfortunately, quadrature rules do not scale well to high dimensions, since multi-dimensional quadrature rules are usually constructed as the tensor product of univariate rules. If a univariate quadrature rule has an error bound of $\mathcal{O}(N^{-r})$, the corresponding rule in d dimensions with N^d integration points would achieve an error bound $\mathcal{O}(N^{-r/d})$, which degrades quickly due to the curse of dimensionality.

Sparse Grids

Sparse grids (Bungartz and Griebel 2004; Smolyak 1960) try to circumvent the scaling issues of standard quadrature methods via a special truncation of the tensor product expansion of univariate quadrature rules. This effectively

reduces the number of integration points to $O(N(\log N)^{d-1})$ without significant drops in accuracy (Gerstner and Griebel 2010). However, even if the underlying quadrature formulas are positive, sparse grids can yield negative weights. In our preliminary experiments, this property of sparse grids was highly problematic, in particular when inference was used as part of a learning routine.

Monte Carlo (MC)

Monte Carlo methods cast the integral as an expectation such that we can compute $\int p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z}$ as $\mathbb{E}_{p(\mathbf{z})}[p(\mathbf{x} \mid \mathbf{z})] = \sum_{i=1}^{N} {}^{1}\!/_{N}\; p(\mathbf{x} \mid \mathbf{z}_i)$. MC is easy to use and especially attractive for high-dimensional problems since its convergence rate $\mathcal{O}(N^{-1/2})$ is not directly dependent on the problem dimensionality. However, this convergence rate decelerates quickly as one increases the number of integration points, which can be too slow.

Quasi-Monte Carlo (QMC) (Caflisch 1998) replaces (pseudo-)random sequences of integration points with low-discrepancy sequences, with the intent of reducing the variance in the MC estimator and converge faster than $\mathcal{O}(N^{-1/2})$. Crucially, QMC is deterministic, which makes it hard to estimate the integration error in practice. Randomised quasi-Monte Carlo (RQMC) reintroduces randomness into the low-discrepancy sequences of integration points, allowing for error estimation (via multiple simulations) and essentially turning QMC into a variance reduction method (l'Ecuyer 2016).

In our experiments, we opt for RQMC for mainly two reasons.

- First, its convergence does not depend directly on the dimensionality, meaning we have more freedom to choose the latent dimensionality. In fact, we empirically observe that increasing the latent dimensionality does not hurt performance (see Section 8.4.2 for an analysis on the influence of the dimension of the latent space). We conjecture that training via numerical integration sufficiently regularises the decoder so that it remains amenable to numerical integration, even when using a relatively large latent dimensionality in the order of tens.

- Second, in contrast to Monte Carlo, it produces integration points of lower variance (l'Ecuyer 2016), which often facilitates training as shown in Section 8.4.2. Moreover, in comparison to QMC, the reintroduction of randomness helps avoid overfitting to a specific set of integration points.

8.3.2 Learning the Decoder

In principle, continuous mixture models can be learnt in many ways, such as amortised inference (Kingma and Welling 2014) or adversarial training (Goodfellow et al. 2014). However, these methods do not encourage the decoder to be amenable to numerical integration, and thus their approximation by (or compilation to) a mixture of tractable models is subpar. Perhaps not surprisingly, we find that training via numerical integration is the best way to learn and extract expressive mixtures of tractable probabilistic models. We compare numerical integration and variational inference in Section 8.4.1 and in Figure 8.2.

Training via numerical integration simply amounts to selecting a set of integration points $\{\mathbf{z}_i\}_{i=1}^N$ using any numerical integration method of choice and, for some training data $\{\mathbf{x}_j\}_{j=1}^M$, maximising the log-likelihood (LL) of the approximate model with respect to the decoder parameters:

$$LL = \sum_{j=1}^{M} \log \sum_{i=1}^{N} \big[w(\mathbf{z}_i)\, p(\mathbf{x}_j \mid \boldsymbol{\theta}(\mathbf{z}_i)) \big]. \tag{8.6}$$

For most numerical integration methods, when $N \to \infty$, this objective converges to the exact log-likelihood of the continuous mixture, and naturally for $1 \ll N < \infty$ it serves as a reasonable approximation.

Specifically, when using (RQ)MC methods the inner sum of Equation 8.6 is unbiased, yielding a *negatively biased* estimate of the true log-likelihood due to Jensen's inequality—i.e. a 'noisy lower bound'—justifying the approximate log-likelihood in Equation 8.6 as training objective for similar reasons as the variational ELBO of Equation 8.5. However, Equation 8.6 should not be confused with the standard ELBO as it does not involve a posterior approximation and, unlike the ELBO, becomes tight for $N \to \infty$. We further note that (RQ)MC methods to estimate the log-likelihood of latent variable models is not a new idea and has been widely explored either directly (MacKay 1995) or to improve ELBO techniques (Burda et al. 2015; Mnih and Rezende 2016; Buchholz et al. 2018). In this chapter, however, we are specifically interested in combining continuous mixtures with *tractable models* via numerical integration, as this direction has been explored rather little.

One can interpret our learnt model in two distinct ways. The first is to interpret it as a 'factory' method, whereby each fixed set of latent variables $\{\mathbf{z}_i\}_{i=1}^N$ yields a tractable model supporting exact likelihood and marginalisation, namely a PC (trivially a mixture of PCs is a PC). The second is to take it as an intractable continuous latent variable model, but one that is amenable to

numerical integration. At test time, we are free to choose the set of integration points, possibly changing the integration method and number of integration points N if more or less precision is needed.

8.3.3 Efficient Learning

When using a neural network to fit $p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))$, computing the backward pass with respect to the log-likelihood objective in (8.6) can be memory intensive. We can circumvent that by first finding the K integration points that are most likely to have generated each training instance. We do so via a forward pass (with no gradient computation) that allows us to identify the K values of $\mathbf{z}$ (among the N points $\{\mathbf{z}_i\}_{i=1}^N$ defined by the integration method) that maximise $w(\mathbf{z})\, p(\mathbf{x}_j \mid \boldsymbol{\theta}(\mathbf{z}))$ for each $\mathbf{x}_j$ in a training batch. We then run backprop to optimise a cheaper estimate of the log-likelihood (8.7) which only requires K values $\{\mathbf{z}_{ij}\}_{i=1}^K$ for each $\mathbf{x}_j$, instead of N

$$LL' = \sum_{j=1}^{M} \log \sum_{i=1}^{K} w(\mathbf{z}_{ij})\, p(\mathbf{x}_j \mid \boldsymbol{\theta}(\mathbf{z}_{ij})). \tag{8.7}$$

For small K this is a crude approximation of the true log-likelihood of the continuous mixture, but we find empirically that it already yields good gradient estimates. Importantly, for $K \ll N$ this results in large improvements in memory efficiency. In our experiments, we only use this approximation for non-binary image datasets for which $K = 1$ was already sufficient to get good results.

8.3.4 Latent Optimisation

As mentioned in the previous sections, once the decoder is trained, we can compile a continuous mixture into a PC by fixing a set of integration points selected by any integration method, leading to a discrete mixture of PCs. However, rather than using a fixed integration scheme, one might also treat the integration points as 'parameters' and optimise them. Specifically, given training instances $\{\mathbf{x}_j\}_{j=1}^M$, we might find suitable $\{\mathbf{z}_i\}_{i=1}^N$ by maximising the log-likelihood:

$$\arg\max_{\{\mathbf{z}_i\}_{i=1}^N} \sum_{j=1}^{M} \log \sum_{i=1}^{N} w(\mathbf{z}_i)\, p(\mathbf{x}_j \mid \boldsymbol{\theta}(\mathbf{z}_i)). \tag{8.8}$$

Due to the similarity to (Bojanowski et al. 2018; Park et al. 2019) we refer to this technique as *Latent Optimisation (LO)*. There are, however, a few differences in spirit. In (Bojanowski et al. 2018; Park et al. 2019) the decoder and individual latent representations, one for each training instance, are jointly learnt to minimise the reconstruction error. The latent space is regularised to follow a Gaussian prior but otherwise is devoid of any probabilistic interpretation. In contrast, in our approach the goal is an accurate yet compact approximation of the true continuous mixture model. Unfortunately, training the decoder and integration points together would lead to overfitting, meaning that the model would not translate well to different integration points or methods. For that reason, we only optimise the integration points as a *post-processing step*, i.e. the *decoder parameters remain fixed* throughout the latent optimisation process.

Our LO approach can also be interpreted as a way to learn (or compile) PCs, using continuous mixtures as a teacher model or regularizer. In our experiments, we see that this approach is remarkably effective. Specifically, LO achieves the same test log-likelihoods as RQMC while using considerably fewer integration points, yielding compacter but still accurate PCs.

8.4 Experiments

We evaluated our methods on common benchmarks for generative models, namely 20 standard density estimation datasets (Lowd and Davis 2010; Bekker et al. 2015; Van Haaren and Davis 2012) as well as 4 image datasets, Binary MNIST (Larochelle and Murray 2011), MNIST (LeCun et al. 1998), Fashion MNIST (Xiao et al. 2017) and Street View House Numbers (SVHN) (Netzer et al. 2011). All models were developed in python 3 with PyTorch (Paszke et al. 2019) and trained with standard commercial GPUs. We used RQMC in all experiments ($w(\mathbf{z}_i) = 1/N$), with samples generated by QMCpy (Choi et al. 2020+). More details on the model and experimental setup as well as extra results are given in Appendix B. The source code for these experiments is available at `github.com/alcorreia/cm-tpm`.

8.4.1 Standard Density Estimation Benchmarks

As a first experiment, we compared *continuous mixtures of factorisations*, $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$, and *continuous mixtures of CLTs*[3], $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$, as density estimators on a series of

[3] CLTs are learnt with the classical algorithm (Chow and Liu 1968) and kept fixed during training.

Table 8.1: Average test log-likelihoods on 20 density estimation benchmarks. We compare $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$, $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ and $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$ with the best performance (BestPC) over 5 PC learning algorithms: Einet (Peharz et al. 2020a), LearnSPN (Gens and Domingos 2013), ID-SPN (Rooshenas and Lowd 2014), RAT-SPN (Peharz et al. 2020b) and HCLT (Liu and Van den Broeck 2021). We train using 2^{10} integration points and we sample 2^{13} mixture components from both $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$. LO is run over 2^{10} integration points.

Dataset	BestPC	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$	$\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$	Dataset	BestPC	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$	$\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$
accidents	**-26.74**	-33.27±0.03	-28.69±0.01	-28.81±0.02	jester	-52.46	**-51.93**±0.02	-51.94±0.01	-51.94±0.02
ad	-16.07	-18.71±0.15	-14.76±0.10	**-14.42**±0.09	kdd	**-2.12**	-2.13±0.00	**-2.12**±0.00	**-2.12**±0.00
baudio	-39.77	**-39.02**±0.02	**-39.02**±0.02	-39.04±0.02	kosarek	-10.60	-10.71±0.01	-10.56±0.01	**-10.55**±0.01
bbc	-248.33	**-240.19**±0.29	-242.83±0.55	-242.79±0.58	msnbc	**-6.03**	-6.14±0.01	-6.05±0.00	-6.05±0.00
bnetflix	-56.27	-55.49±0.02	**-55.31**±0.02	-55.36±0.02	msweb	-9.73	-9.68±0.00	-9.62±0.01	**-9.60**±0.01
book	-33.83	-33.67±0.04	-33.75±0.03	**-33.55**±0.02	nltcs	**-5.99**	**-5.99**±0.00	**-5.99**±0.01	**-5.99**±0.00
c20ng	-151.47	-148.24±0.10	**-148.17**±0.09	-148.28±0.11	plants	-12.54	-12.45±0.02	**-12.26**±0.10	-12.27±0.01
cr52	-83.35	-81.52±0.08	**-81.17**±0.11	-81.31±0.15	pumbs	**-22.40**	-27.67±0.03	-23.71±0.03	-23.70±0.02
cwebkb	-151.84	-150.21±0.22	-147.77±0.26	**-147.75**±0.26	tmovie	-50.81	**-48.69**±0.09	-49.23±0.10	-49.29±0.12
dna	**-79.05**	-95.64±0.37	-84.91±0.09	-84.58±0.10	tretail	-10.84	10.85±0.00	-10.82±0.01	**-10.81**±0.01
Avg. rank	2.85	2.65	1.85	**1.75**					

20 standard commonly used benchmark datasets. In this set of experiments, we fixed the mixing distribution $p(\mathbf{z})$ to a 4-dimensional standard Gaussian.

We ran our models on each dataset using 5 different random seeds and trained for up to 200 epochs, with batch size 128, employing early stopping on the validation set with patience 15 to avoid overfitting. As decoders we used 6-layer MLPs with a progressively increasing number of units, LeakyReLUs activations, and batch normalisation layers (Ioffe and Szegedy 2015). For every optimisation step, we constructed a new set of integration points $\{\mathbf{z}_i\}_{i=1}^N$ from randomised lattice sequences of size $N = 2^{10}$. We then maximised the log-likelihood as in Equation 8.6 using Adam (Kingma and Ba 2014).

At test time, the trained models can be evaluated with any number of integration points N, yielding a mixture of PCs and consequently indeed a standard PC (Vergari et al. 2020). Table 8.1 reports the test log-likelihoods for 2^{13} components[4], averaged over the 5 random seeds, and state-of-the-art (SOTA) competitors. Our results set SOTA log-likelihoods for tractable models on 16 out of 20 datasets and are competitive on the remaining 4. For each dataset, we ranked the performance of the models involved from 1 to 4, and reported the average rank at the bottom of the first half of the table. In particular, we notice a substantial gap in performance between $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ for the datasets

[4]This was the largest number of components we used, which seemed sufficient for all datasets, as the results did not change significantly with additional integration points. For some datasets, the continuous model is already approximated well with much smaller mixtures. See Appendix B.

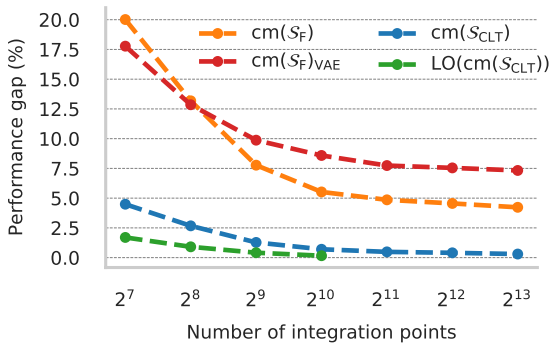

Figure 8.2: Relative performance gap to the best test log-likelihood in Table 8.1 as a function of the number of integration points and averaged over all 20 datasets. Latent Optimisation (LO) is run (on purpose) for fewer number of integration points yet performs best. Lower is better.

`accidents`, `ad`, `dna` and `pumbs`, which are known to be highly structured. We want to stress that we used the same hyperparameters for all datasets, and thus our SOTA results do not stem from extensive tuning efforts.

Figure 8.2 shows the effect of the number of integration points at test time, by plotting the performance of our models relative to the best results in Table 8.1. In this plot of Figure 8.2, we aggregated the results by averaging across all 20 datasets, but in Figure B.1 in Appendix B we show the same curves for each individual dataset. Note that $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ generally outperforms $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and the gap is accentuated when few integration points are used.

Latent Optimisation

We also investigated the effect of Latent Optimisation (LO) and learnt the integration points after having fit the decoder, as discussed in Section 8.3.4. Specifically, we run LO to search for a set of integration points for our trained $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ by maximising (8.8) and show the results under $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$ both in Table 8.1 and Figure 8.2, where we compare LO against different continuous mixtures.

We see that $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$ achieves essentially the same performance as $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ but with *8 times fewer integration points,* leading to much smaller PCs. However, as can be seen in Figure 8.2, for a large number of integration points, (RQ)MC estimates already have low-variance and there is little room for

improvement with LO. In fact, in this setting LO is prone to overfitting. This can be expected when the number of integration points becomes too large (in comparison to the number training data points), since they are treated as trainable parameters. For that reason, we limit our LO experiments to 2^{10} integration points in all datasets in Table 8.1 and 8.4.

Comparison to Variational Learning

As the main idea in this chapter is connecting continuous mixtures with PCs via numerical integration, we have used Equation 8.6 to train our models thus far. However, the processes of training a continuous mixture model and of converting it into a PC are orthogonal to each other, so we might as well use traditional VAE training to learn continuous mixtures, i.e. maximising the ELBO (Equation 8.5) instead. In other words, we can train vanilla VAEs with a *small latent dimensionality* and convert the resulting models into PCs using RQMC or LO.

Table 8.2: Test log-likelihoods for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ with 4 latent dimensions and trained for 200 epochs on 20 binary density estimation benchmarks. We compute the ELBO with 1000 MC samples and report the mean and standard deviation of results obtained with 5 different random seeds.

	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$								$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$
Dataset	ELBO	Mix-2^7	Mix-2^8	Mix-2^9	Mix-2^{10}	Mix-2^{11}	Mix-2^{12}	Mix-2^{13}	Mix-2^{13}
accidents	-34.56±0.19	-37.34±0.62	-35.78±0.23	-35.25±0.12	-34.80±0.15	-34.61±0.25	-34.59±0.27	-34.55±0.25	**-33.27**±0.03
ad	-22.27±0.92	-38.39±2.80	-30.25±2.25	-25.35±1.31	-23.18±1.43	-21.69±0.96	-21.36±0.96	-21.08±1.02	**-18.71**±0.15
baudio	-39.36±0.20	-40.30±0.23	-40.00±0.15	-39.80±0.10	-39.73±0.11	-39.70±0.11	-39.69±0.11	-39.68±0.11	**-39.02**±0.01
bbc	-243.04±0.26	-247.25±0.50	-244.68±0.29	-243.10±0.43	-242.26±0.41	-241.59±0.42	-241.38±0.44	-241.24±0.41	**-240.19**±0.29
bnetflix	-56.38±0.35	-57.30±0.10	-56.97±0.12	-56.78±0.21	-56.74±0.24	-56.72±0.25	-56.71±0.26	-56.71±0.26	**-55.49**±0.02
book	-34.56±0.55	-34.96±0.34	-34.83±0.43	-34.79±0.44	-34.73±0.46	-34.70±0.48	-34.68±0.48	-34.68±0.48	**-33.67**±0.04
c20ng	-149.53±0.32	-152.59±0.88	-151.22±0.40	-150.38±0.13	-149.93±0.22	-149.54±0.24	-149.38±0.30	-149.26±0.32	**-148.24**±0.10
cr52	-82.62±0.22	-86.08±0.57	-84.40±0.30	-83.05±0.22	-82.40±0.18	-81.94±0.10	-81.80±0.11	-81.66±0.10	**-81.52**±0.08
cwebkb	-151.44±0.45	-153.94±0.34	-152.76±0.09	-152.04±0.28	-151.63±0.29	-151.32±0.43	-151.20±0.44	-151.11±0.47	**-150.21**±0.22
dna	-96.77±0.29	-97.25±0.20	-97.06±0.24	-96.79±0.30	-96.69±0.33	-96.59±0.36	-96.56±0.37	-96.53±0.38	**-95.64**±0.37
jester	-52.79±0.33	-53.26±0.25	-53.04±0.21	-53.00±0.24	-52.99±0.25	-52.98±0.25	-52.98±0.25	-52.98±0.25	**-51.93**±0.02
kdd	**-2.05**±0.01	-2.17±0.01	-2.17±0.01	-2.16±0.01	-2.16±0.01	-2.15±0.01	-2.15±0.01	-2.14±0.00	-2.13±0.00
kosarek	-10.75±0.04	-11.07±0.04	-11.00±0.04	-10.98±0.04	-10.96±0.04	-10.96±0.04	-10.96±0.04	-10.96±0.04	**-10.71**±0.01
msnbc	-6.32±0.00	-6.49±0.00	-6.49±0.00	-6.49±0.00	-6.49±0.00	-6.49±0.00	-6.49±0.00	-6.49±0.00	**-6.14**±0.01
msweb	**-9.61**±0.04	-10.17±0.10	-10.08±0.06	-10.01±0.04	-9.99±0.04	-9.98±0.04	-9.98±0.04	-9.98±0.04	-9.68±0.00
nltcs	-6.19±0.16	-6.35±0.12	-6.35±0.12	-6.35±0.12	-6.35±0.12	-6.35±0.12	-6.35±0.12	-6.35±0.12	**-5.99**±0.00
plants	-12.80±0.07	-14.66±0.68	-13.80±0.45	-13.31±0.07	-13.14±0.03	-13.03±0.07	-13.00±0.08	-12.99±0.08	**-12.45**±0.02
pumbs	-29.21±0.37	-37.79±0.92	-33.54±0.47	-30.77±0.18	-29.78±0.19	-29.25±0.32	-29.11±0.39	-28.86±0.35	**-27.67**±0.03
tmovie	-49.01±0.34	-52.07±0.87	-50.77±0.42	-49.83±0.08	-49.43±0.20	-49.15±0.25	-49.07±0.27	-48.98±0.28	**-48.69**±0.09
tretail	**-10.79**±0.05	-11.01±0.01	-11.00±0.01	-11.00±0.01	-11.00±0.01	-11.00±0.01	-11.00±0.01	-11.00±0.01	-10.85±0.00

For this purpose, we stuck to the same experimental setup for the 20 datasets and learnt continuous mixtures with VAE training (Kingma and Welling 2014), denoted as $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$. Note that we used exactly the same architecture and four-dimensional latent space for both $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$, with the only

difference between these models being the training method. We report the results in Table 8.2 and Figure 8.2, where we see $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ performs subpar to training via numerical integration, even with a large number of integration points at test time. We propose two explanations for this result:

- It might be that $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ models are less amenable to numerical integration and that their true log-likelihood is actually higher, or conversely, that models *trained* with numerical integration are also more amenable to numerical integration at test time. Indeed, in Table 8.2 we see that, in 11 out of the 20 datasets, we could not match the ELBO (computed with 1000 MC samples) even when approximating $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ with 2^{13} points.
- It is possible numerical integration leads to better model training for *small latent dimensionality*. We see evidence for that in Table 8.2, since in all but 3 datasets, a cm model trained via numerical integration ($\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$), outperformed $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ in all cases (ELBO and numerical integration).

However, evidently traditional VAE training is superior for large latent dimensionality, as numerical integration degrades quickly in high dimensional spaces. Moreover, variational inference is more efficient than numerical integration: for each data point $\mathbf{x}$, variational inference computes $p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i))$ for only a few samples $\mathbf{z}_i$ from the posterior—even as few as one (Kingma and Welling 2014)—whereas numerical integration has to compute it once for every integration point. Therefore, it would be interesting to be able to extract good mixtures from models learnt via variational inference. We believe one can improve on this by sufficiently regularising the decoder, e.g. by penalising high Lipschitz constants (Arjovsky et al. 2017; Gulrajani et al. 2017), and thus facilitating numerical integration, but we leave that for future research.

Plain Mixture Models

For MC integration with a fixed number of integration points N, the approximate model resembles a plain mixture model with equally-probable weights:

$$\hat{p}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i)) \approx p(\mathbf{x}) = \int p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}))p(\mathbf{z})d\mathbf{z}.$$

Thus it makes sense to compare our models to plain mixture models. That is, instead of having mixture components as a function of some latent variable,

each component $p(\mathbf{x} \mid \boldsymbol{\theta}_i)$ is fully independent with

$$p_{\text{mix}}(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} p(\mathbf{x} \mid \boldsymbol{\theta}_i).$$

In Table 8.3, we compare plain mixtures with 1024 components to a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ with 4 latent dimensions. We use a fully factorised structure for both models and train them using the exact same protocol of previous experiments. The results in Table 8.3 show that our model outperformed plain mixtures, denoted $\mathrm{dm}(\mathcal{G}_F)$, in all datasets, even though both models were trained the exact same way and have the exact same structure if we consider a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ compiled to a 1024 components mixture. The only difference is how we parametrise the mixture components. Similarly, we also considered mixture models with learnable weights, that is

$$p_{\text{mix}}(\mathbf{x} \mid \boldsymbol{\theta}_i) = \sum_{i=1}^{N} w_i p(\mathbf{x} \mid \boldsymbol{\theta}_i),$$

where $\{w_i\}_{i=1}^{N}$ are learnable parameters with $w_i \geq 0$ and $\sum_i w_i = 1$. We evaluated mixture models with learnable weights trained both via gradient descent, denoted $\mathrm{dm}(\mathcal{G}_F^W)$, and Expectation Maximisation, denoted $\mathrm{dm}(\mathcal{G}_F^{EM})$.

As shown in Table 8.3, our model outperforms plain mixtures in all datasets but `dna` and `msnbc`, where learning via EM produces better test log-likelihoods.

Table 8.3: Test log-likelihoods for plain mixtures with non-learnable equally-probable weights trained via gradient descent ($\mathrm{dm}(\mathcal{G}_F)$), with learnable weights also trained via gradient descent ($\mathrm{dm}(\mathcal{G}_F^W)$), with learnable weights trained via EM ($\mathrm{dm}(\mathcal{G}_F^{EM})$), and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ on 20 standard density estimation benchmarks. All models are trained for up to 300 epochs using early stopping and employing 1024 components. Higher is better.

Dataset	$\mathrm{dm}(\mathcal{G}_F)$	$\mathrm{dm}(\mathcal{G}_F^W)$	$\mathrm{dm}(\mathcal{G}_F^{EM})$	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$	Dataset	$\mathrm{dm}(\mathcal{G}_F)$	$\mathrm{dm}(\mathcal{G}_F^W)$	$\mathrm{dm}(\mathcal{G}_F^{EM})$	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$
accidents	-42.58	-40.61	-35.38	**-33.94**	jester	-55.32	-53.54	-52.54	**-52.03**
ad	-104.57	-97.79	-24.91	**-20.42**	kdd	-6.81	-2.15	-2.14	**-2.13**
baudio	-42.24	-40.41	-39.76	**-39.14**	kosarek	-16.20	-11.17	-10.88	**-10.75**
bbc	-281.88	-288.31	-252.82	**-241.54**	msnbc	-6.36	-6.12	**-6.03**	-6.15
bnetflix	-58.19	-57.00	-56.34	**-55.71**	msweb	-18.29	-11.36	-10.00	**-9.72**
book	-41.72	-35.61	-34.66	**-33.79**	nltcs	-6.16	-6.01	-6.00	**-6.00**
c20ng	-163.04	-157.80	-151.79	**-149.10**	plants	-16.66	-14.41	-13.44	**-12.65**
cr52	-104.91	-98.79	-87.07	**-82.33**	pumbs	-46.59	-42.90	-32.84	**-28.50**
cwebkb	-176.60	-170.90	-154.75	**-151.00**	tmovie	-66.94	-61.64	-52.80	**-49.12**
dna	-101.93	-98.14	**-94.46**	-96.11	tretail	-18.35	-11.42	-10.90	**-10.85**

However, existing PC architectures (namely HCLT and RAT-SPNs) already performed better than our models in these datasets, so it is not that surprising that a discrete mixture works well for these two datasets.

For a fixed number of integration points (mixture components) discrete mixture models are strictly more expressive than continuous ones, since in discrete mixtures the parameters of each component are completely independent. Yet, continuous mixtures outperformed discrete ones in almost all cases, indicating the regularisation introduced by the shared 'decoder' does facilitate learning. This result, albeit not surprising, has never been fully exploited in the tractable probabilistic models literature, to the best of our knowledge.

8.4.2 Binary MNIST

Table 8.4: Binary-MNIST test log-likelihoods for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ for different numbers of integration points at test time N. Both models were trained with 2^{14} integration points. We also report latent optimisation results, i.e. $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{F}}))$ and $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$, for up to 2^{10} integration points. Higher is better.

	Model			
N	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{F}}))$	$\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$	$\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$
2^7	-167.29	-144.00	-127.59	-114.02
2^8	-150.67	-135.89	-119.09	-110.02
2^9	-138.55	-129.15	-113.15	-107.14
2^{10}	-129.24	-123.44	-108.30	-104.37
2^{11}	-121.96	-	-104.50	-
2^{12}	-116.42	-	-101.55	-
2^{13}	-112.03	-	-99.23	-
2^{14}	-108.69	-	-97.48	-

We further evaluated our models on Binary MNIST (Larochelle and Murray 2011). We followed the same experimental protocol as in the previous experiments, except that we employed a larger latent dimensionality of 16 and therefore increased the number of integration points during training to 2^{14}. We did *not* use convolutions but rather stuck to the 6-layer MLP architecture scheme. We ran $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ and applied LO to both final models for up to 50 epochs, using early stopping on the validation set to avoid overfitting. Table 8.4 shows that overall $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ outperforms $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and that Latent Optimisation is remarkably effective when relatively few integration points are used.

Effect of Latent Space Dimensionality

We also investigated the effect of the latent space dimensionality on the overall performance of the model. We trained $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ models on Binary MNIST varying the latent space dimension d, with $d = 2^i$ for $i \in \{0, 1, 2, 3, 4, 5, 6\}$. We train all models with 2^{14} integration points but vary the number of integration points at test time from 2^8 to 2^{16}, as shown in Figure 8.3. In the following discussion, when referring to integration points we mean those used at *test time*.

Table 8.5: Test log-likelihoods on Binary MNIST for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ with varying latent dimensionality. ELBO computed with 1000 Monte Carlo samples and log-likelihoods (LL) computed via RQMC with 2^{16} points. Higher is better.

Latent dimensions	1	2	4	8	16	32	64
ELBO (VAE)	-154.13	-132.21	-112.77	**-97.26**	**-93.66**	**-92.24**	**-92.74**
LL ($\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$)	-148.87	-127.57	-111.57	-110.67	-114.76	-115.36	-116.57
LL ($\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$)	**-128.05**	**-112.60**	**-106.97**	-103.82	-104.08	-104.29	-104.21

For low numbers of integration points (up to 1024), models with small latent spaces perform best. That is as expected, since small latent spaces are more amenable to numerical integration. However, as we increase the number of integration points, it is clear the model benefits from larger latent dimensions, with improvements in performance up until 8 latent dimensions and only relatively small deterioration thereafter. That is somewhat surprising, since numerical integration in high dimensions is notoriously difficult and we would expect performance to deteriorate much more drastically as we increase the latent dimensionality. We hypothesise training via numerical integration strongly regularises the decoder so that the learnt function is smooth enough to allow for reliable numerical integration irrespective of the latent dimensionality.

In Figure 8.3, we report the same analysis for cm models trained via variational inference, $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$. We used the same decoder architecture and training protocol as for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$. Here we do observe that numerical integration struggles with high latent spaces, as test log-likelihoods rapid decline for $d \geq 4$. That is simply because numerical integration becomes harder, since the ELBO actually improves for higher latent dimensions as shown in Table 8.5. In all cases, it seems easier to numerically integrate—or extract good mixtures of tractable models from—models learnt via numerical integration than models trained via variational inference. For $d \leq 4$, numerical integration outperforms variational inference, whereas for $d > 4$ we still get better mixtures from $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ than from $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$, although the latter is more expressive as indicated by the ELBO.

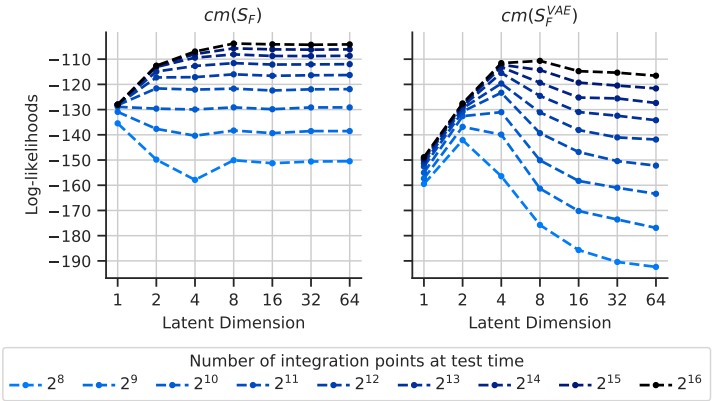

Figure 8.3: Test log-likelihood on Binary MNIST against latent dimensionality of $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ evaluated with different numbers of integration points.

Monte Carlo (MC) vs Randomised Quasi Monte Carlo (RQMC)

We also used Binary MNIST data to compare the performance of Monte Carlo (MC) and Randomised Quasi Monte Carlo (RQMC) as numerical integration methods for training continuous mixtures.

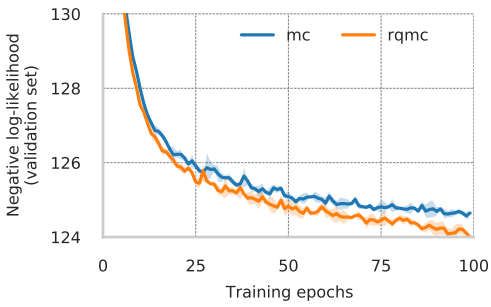

Figure 8.4: Log-likelihood on validation set against number of training epochs for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ trained on Binary MNIST with MC (blue) and RQMC (orange). We use 2^{10} integration points in both cases and plot one standard deviation confidence bands computed with 5 different random seeds.

In practice, Randomised Quasi Monte Carlo (RQMC) can be seen as a variance reduction method (l'Ecuyer 2016), which often translates into faster convergence of learning algorithms (Buchholz et al. 2018). We also have observed RQMC to improve convergence in our use cases, and thus have used RQMC in all our experiments. To illustrate this, we compare the training of $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ on Binary MNIST using MC and RQMC. In Figure 8.4, it is clear that RQMC facilitates learning and improves the overall solution by a small but non-negligible amount.

8.4.3 Image Datasets

We further evaluated our methods on (non-binary) image datasets. This time we use a convolutional architecture similar to that of DCGAN (Radford et al. 2015), but we also included residual convolutional blocks as in (Van Den Oord et al. 2017). We used a similar training protocol as the one described for binary datasets, maintaining a maximum of 300 epochs and early-stopping with patience of 15, but increasing the batch size to 512. As for Binary MNIST, we used 2^{14} integration points during training for all image datasets. We compare against Einsum Networks (or EiNets) (Peharz et al. 2020a), large scale PCs specifically designed to take advantage of GPU accelerators.

Modelling Images as Discrete Data

We start of by considering images as discrete data, i.e. we compute densities directly in the 1-bit pixel space for Binary MNIST, in which case the leaves of each PC, both Einets and our models, define Bernoulli distributions; and in the 8-bit pixel space for non-binary image data, which is modelled by 256-dimensional categorical distributions at the leaves. In this last scenario, we relied on the approximation introduced in Section 8.3.3 with $K = 1$ since computing the loss in Equation 8.6 with 256 parameters per pixel is extremely demanding in terms of memory requirements.

We experimented with two Einets of different sizes for each dataset. For Binary MNIST, 'Small Einet' and 'Big Einet' had respectively 5 and 84 million parameters; for MNIST, 11 and 90 million parameters; and for SVHN, 28 and 186 million parameters. The architectures of both Einets were defined via the Poon-Domingos structure scheme (Poon and Domingos 2011), recursively dividing the images in contiguous square blocks.

For all models, we fed the training data to the model without pre-clustering it into different components of the root sum node. Such a pre-processing step is

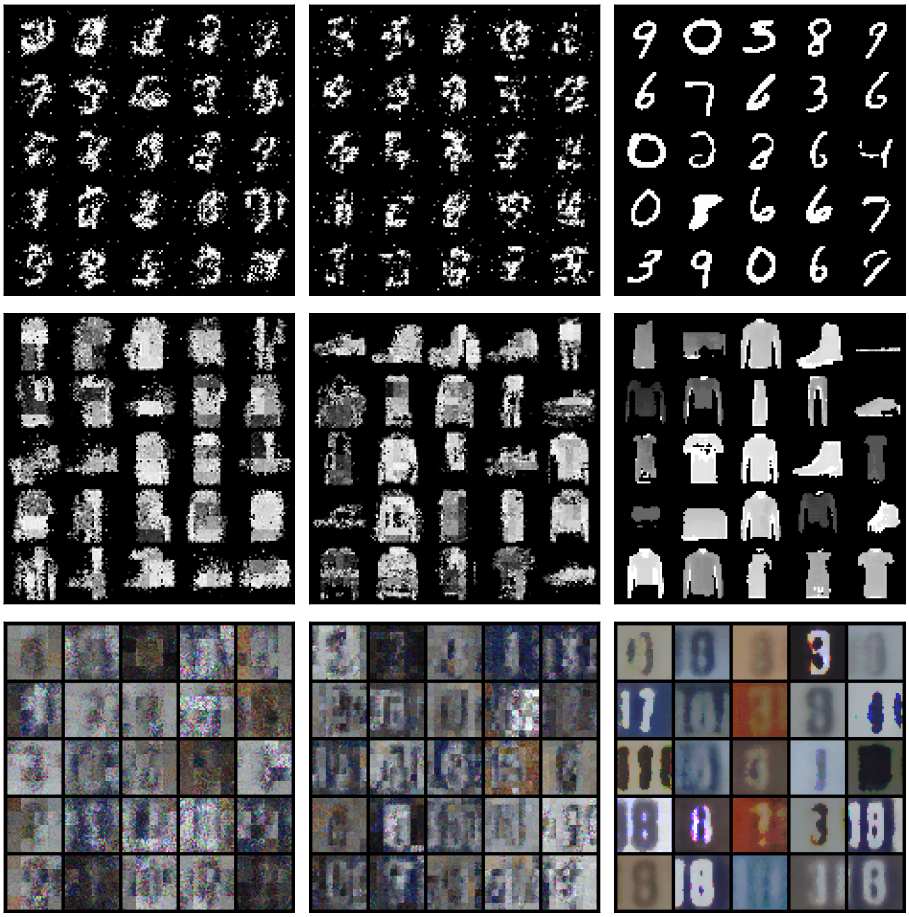

Figure 8.5: Image samples from models with *256-dimensional categorical distributions* at the leaves: 'Small Einet' (left column), 'Big Einet' (middle column) and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ (right column). We can see $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ samples are smoother and suffer less from pixelation issues in comparison to images generated by Einets.

applicable to any method and does not add to the analysis in this chapter. That is why sample quality of Einets observed in Figure 8.5 is worse than as reported in (Peharz et al. 2020a), where images were clustered and a dedicated Einet was trained on each cluster.

As seen in Table 8.6, continuous mixtures outperform EiNets in all image datasets but SVHN in the discrete data case. That is remarkable since our models are extremely compact with the decoder given by a light convolutional architecture of approximately 100K free parameters for MNIST data, and 300K for SVHN; orders of magnitude smaller than the competing EiNets. Moreover, our models also achieve better sample quality. In Figure 8.5, we see samples from $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ are clearly sharper and do not suffer from intense pixelation like those from EiNets. We conjecture that one of the main reasons for the better sample quality of $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ is the absence of product nodes in its architecture, simply because the children of a product undergo the sampling procedure independently. This is of course mitigated by sophisticated PC architectures, where each product node has a rich context defined by its parents, but is still visible in samples from Poon-Domingos architectures like the ones we used in our experiments. In Figure 8.5, we can clearly see the square blocks into which the image space is partitioned in our Einets.

Table 8.6: Bits per dimension (bpd) for image data and models using categorical distributions at the leaves. Lower is better.

Model	'Small Einet'	'Big Einet'	Ours (2^{14})
Binary-MNIST	0.206	0.184	**0.179**
MNIST	1.490	1.415	**1.282**
Fashion-MNIST	3.938	3.737	**3.546**
SVHN	6.442	**5.961**	6.307

The comparatively worse performance of our model on SVHN could be due to the number of integration points. We used 2^{14} for both MNIST and SVHN, but it is reasonable to assume the latter requires a more sophisticated latent representation, and hence higher numbers of integration points at training time. Unfortunately, exploring higher numbers of integration points would be too computationally expensive for our compute infrastructure, and we leave that for future work. In fact, this showcases an important limitation of out method, which is its computational cost. Scaling our continuous mixtures to larger datasets will likely require new learning techniques beyond the 'shortcut' discussed in Section 8.3.3. Other numerical integration methods or regularised variational

objectives similar to that of VAEs might play a role in enabling more complex latent spaces that are still well approximated by discrete mixtures.

Modelling Images as Continuous Data

Directly modelling discrete data with continuous distributions is problematic, since one can achieve arbitrarily high log-likelihoods simply by concentrating the density mass in narrow regions around each of the possible discrete values that pixels can assume, $\{0, \ldots, 255\}$. Therefore, following best practices from the literature in generative models (Uria et al. 2013; Theis et al. 2015), we employed a 'jittering' process whereby one adds uniform noise to pixel values and subsequently divides the result by 256. This procedure is also theoretically justified as the log-likelihoods thus obtained on continuous data are closely related to the log-likelihoods on the original discrete data (Theis et al. 2015).

In this setting, we used the same architectures and training protocol from previous experiments with $K = 1$ (see Section 8.3.3), but this time the leaves were parametrised as normal distributions with learnable mean and variance. To avoid numerical issues, we bounded variance values between $1\mathrm{e}{-6}$ and 1.0 during training. This parametrisation results in slightly more compact models: for MNIST, 'Small Einet' and 'Big Einet' had respectively 5 and 84 million parameters; and for SVHN, 5 and 163 million parameters.

Table 8.7: Bits per dimension (bpd) for image data and models using normal distributions at the leaves. Lower is better.

Model	'Small Einet'	'Big Einet'	Ours (2^{14})
MNIST	2.651	**2.057**	2.762
Fashion-MNIST	4.309	**3.938**	4.485
SVHN	6.332	**5.972**	6.400

Interestingly, in that scenario our model $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ is outperformed by both Einets. One possible explanation is that the optimisation problem is intrinsically more complicated in the continuous case, with the variance of normal distributions being particularly hard to optimise (Dai and Wipf 2019). Einets mitigate that by using online EM (Peharz et al. 2016, 2020a), which performs a form of natural gradient descent (Sato 1999), and thus greatly facilitates learning (Amari 1998). Still, there remains a gap in performance between PCs learnt on discrete data (Table 8.6) and PCs learnt on continuous data (Table 8.7). Curiously, that gap is much smaller on SVHN than on MNIST, possibly because it is

easier to scale normal distributions, which require learning only two parameters per leaf in comparison to the 256 parameters of categorical distributions.

In terms of sample quality, continuous mixtures still generate smoother and less pixelated images than Einets as seen in Figure 8.6. That is a clear manifestation that sample quality and log-likelihood values are weakly correlated (Theis et al. 2015). Moreover, by comparing Figures 8.5 and 8.6, we can also conclude leaves parameterised with normal distributions produce images that are smoother and better looking but also more blurry.

Nonetheless, there is an important caveat in the variability of samples generated by continuous mixtures with either normal or categorical leaves. Consider the PC that is induced by the numerical integration of a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ with N integration points. Such a PC is essentially a shallow mixture model and is thus only capable of generating N meaningfully different samples. That is because in any given mixture component, any variation in the generated samples comes from univariate distributions at the leaves which only amounts to some added noise that is visible in the images of Figures 8.5 and 8.6. That is particularly apparent in the case of continuous mixtures with normal distributions, for which we can plot only the mean values of the sampled leaves, ignoring the variance of individual pixel values. We can see in Figure 8.7 that images thus generated are more visually appealing, suggesting the variance at the leaves is not particularly useful for sample generation.

Naturally, a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ might be capable of capturing more variance in the data and, in the context of this experiment, represent a larger collection of images. That is, we can get a more diverse set of samples if we resample $\mathbf{z} \sim p(\mathbf{z})$ each time, instead of using a compiled PC approximating the underlying $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$. Unfortunately, if we rely on the continuous latent space for sampling, we are not, strictly speaking, sampling from a tractable model, as we must rely in approximate numerical integration methods to compute likelihoods. As mentioned previously, these can be made arbitrarily precise, but that is still not directly comparable to fully tractable models like Einets. At any rate, the comparisons in Figures 8.5, 8.6 and 8.7 remain valid (we sample from a compiled PC), although we recognise they are not enough to fully evaluate the variance of the images generated by each model.

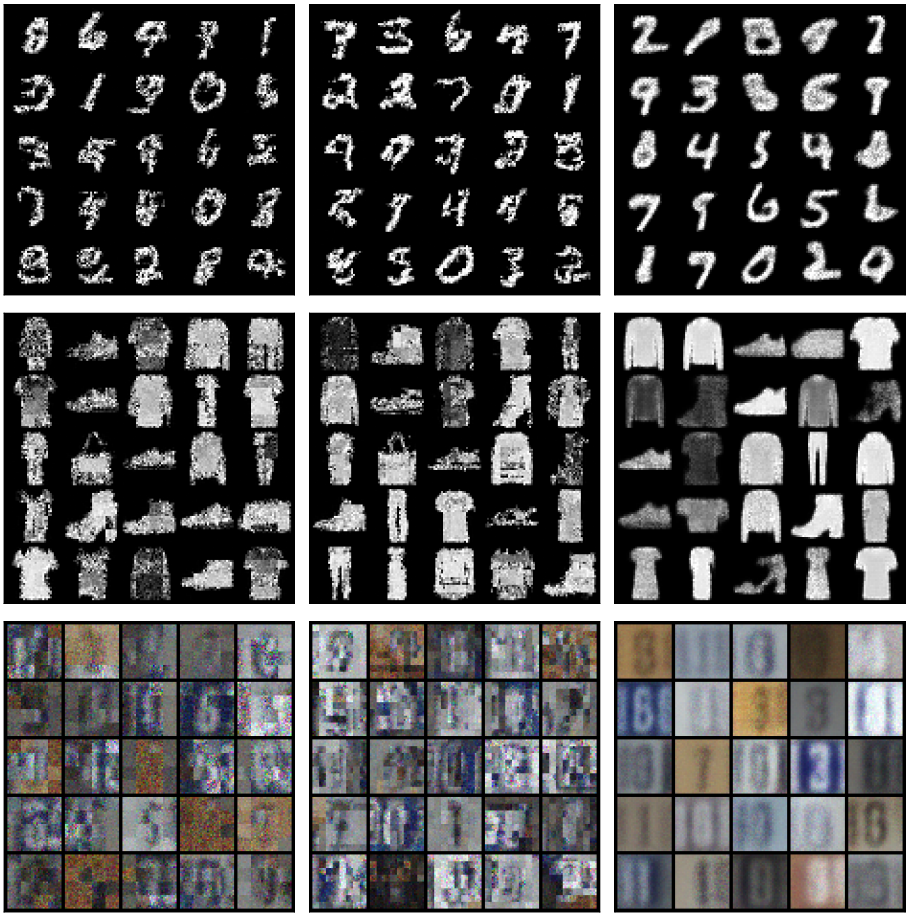

Figure 8.6: Image samples from models with *normal distributions* at the leaves: 'Small Einet' (left column), 'Big Einet' (middle column) and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ (right column). Once more we see $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ offers better sample quality than Einets.

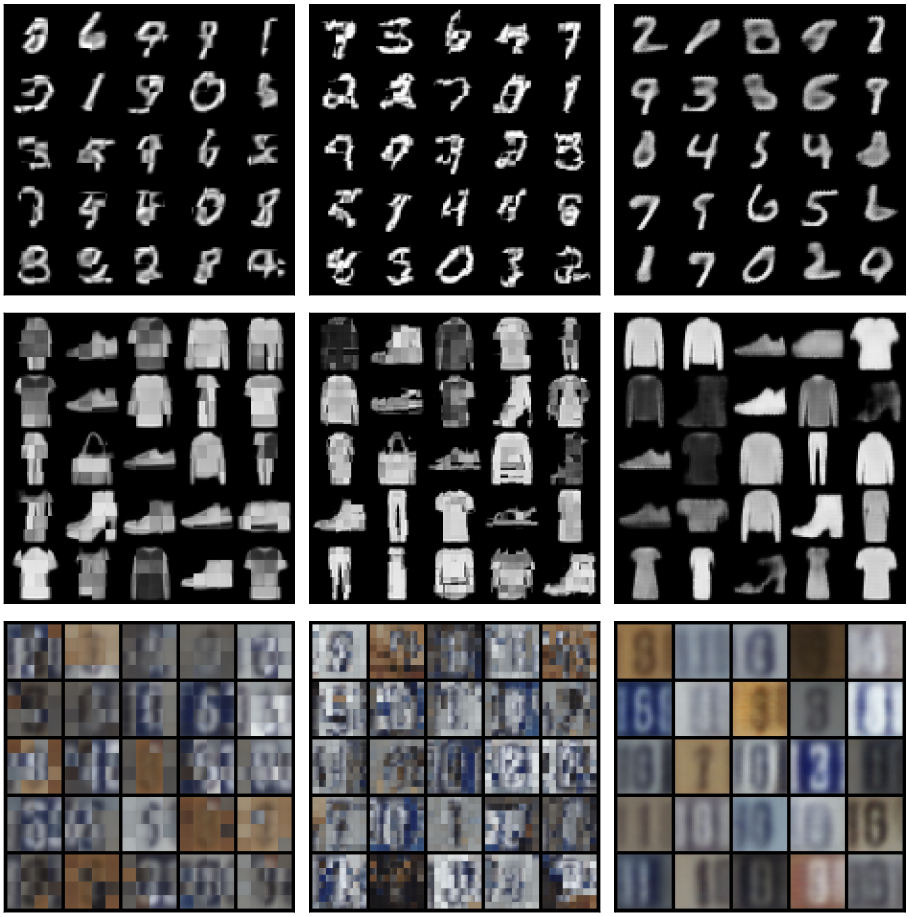

Figure 8.7: Image samples from models with *normal distributions* at the leaves but ignoring the variance of individual pixels: 'Small Einet' (left column), 'Big Einet' (middle column) and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ (right column).

8.4.4 Other Tractable Queries

Our models also support efficient marginalisation, since the discrete approximation obtained via numerical integration is a PC in itself. That allows us to handle missing data and perform tasks like inpainting out-of-the-box, without any extra modelling steps. While this is not the focus of our work—these queries are well-established for PCs, and it is not surprising that our models support them as well—it is interesting to see how our models perform in those tasks. For this we used image data and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ models with Bernoulli (Binary MNIST) or normal distributions at the leaves (MNIST and Fashion-MNIST).

We successfully trained our model on Binary MNIST with substantially parts of the data missing. Note that such a training procedure is delicate for intractable models like VAEs. Furthermore, we included inpainting experiments on Binary MNIST, MNIST and Fashion-MNIST, i.e. reconstructing missing data at test time, using a model trained on complete data.

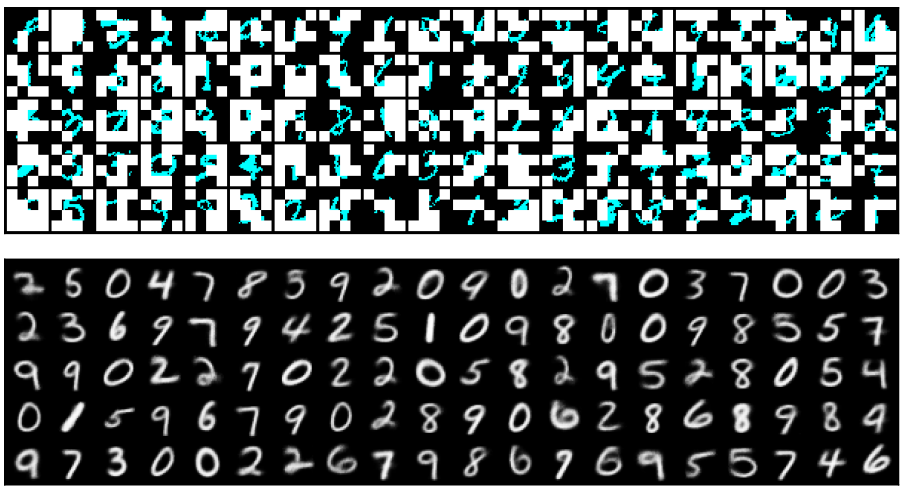

Figure 8.8: Illustration of the ability of $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ to handle missing data via marginalisation. On top we have Binary MNIST training images with blocks of pixels omitted at random. We display the training images in colour to distinguish pixel values from non-observed pixels (depicted in white). On the bottom, we have samples (continuous Bernoulli) from a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ trained on incomplete Binary MNIST images, like the ones shown above. The models always generates complete digits, despite never observing one during training.

Training with missing data

Our models support efficient marginalisation out-of-the-box both at training and test time, like any other Probabilistic Circuit. More precisely, for any partition of the domain variables into observed $\mathbf{X}_o$ and not observed variables $\mathbf{X}_{\neg o}$, one can compute $p(\mathbf{X}_o) = \int_{\mathbf{x}_{\neg o}} p(\mathbf{X}_o, \mathbf{x}_{\neg o}) d\mathbf{x}_{\neg o}$, with the same computational complexity of computing $p(\mathbf{X})$. This is formally demonstrated for Sum-Product Networks (Peharz et al. 2015); a class of PCs to which our approximate model $\sum_{i=1}^{N} w(\mathbf{z}_i) p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i))$ belongs to.

In a nutshell, marginalisation in PCs boils down to evaluating marginals at the leaves, which is often simple to compute (e.g. with univariate leaves). That is the same for continuous mixtures, where inference is performed via the approximate model above; itself a PC supporting efficient exact marginalisation. That means, numerical integration can approximate any marginal query in continuous mixtures in the exact same principled manner, without requiring imputation or ad-hoc techniques to handle missing values. We illustrate this by showing samples from a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ trained only with incomplete Binary MNIST samples, as depicted in Figure 8.8. At training time, we omit blocks of pixels from each image completely at random, meaning the model never observes a complete digit. Nonetheless, the model generates complete and reasonable digits, indicating proper marginalisation of the missing pixels at training time.

Inpainting

Our models also support efficient computation of *approximate* most probable explanations (MPE) queries. This property also stems directly from the underlying PC in the approximate model $\sum_{i=1}^{N} w(\mathbf{z}_i) p(\mathbf{x} \mid \boldsymbol{\theta}(\mathbf{z}_i))$. In Figure 8.9, we demonstrate this type of query via inpainting with a $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ model on MNIST data. To compute MPE for the images in Figure 8.9, we first compile the model to a discrete mixture with 2^{14} integration points and evaluate the approximate model on the incomplete images (possible thanks to efficient marginalisation in PCs). That allows for a simple (approximate) posterior inference whereby we select, for each image, the component (integration point) most likely to have generated it. Finally, we take the most likely value from these individual components to get the final reconstruction. Albeit simple, this MPE procedure produces good reconstructions as shown in Figure 8.9. Note that we used the same set of integration points to run MPE on each and every image in Figure 8.9. That means these results are compatible with both interpretations of the model: numerical integration on the continuous mixture, or exact inference on a compiled PC.

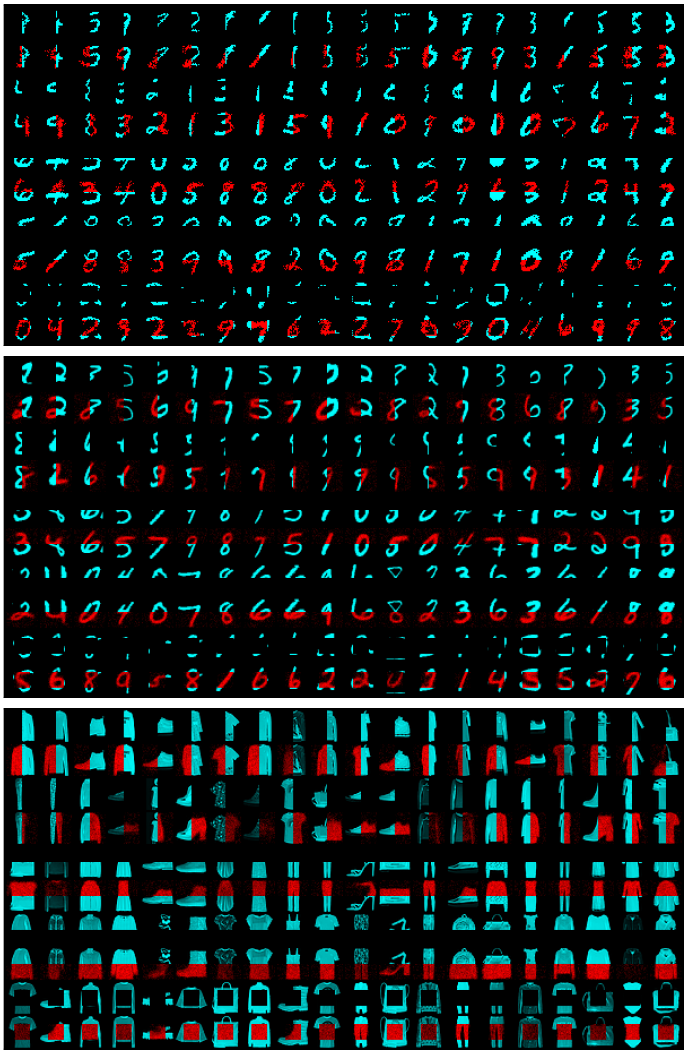

Figure 8.9: Inpainting results on Binary MNIST (top), MNIST (middle) and Fashion-MNIST (bottom) obtained with $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$. In alternating rows, we have first the original image with a missing part, and below the reconstructed image output by $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$. Pixels from the original (reconstructed) images are depicted in blue (red).

8.5 Discussion

Our experiments demonstrate that continuous mixtures of PCs—or actually their discrete approximations yielding again PCs—outperform most previous PC methods on several datasets. At first this might appear surprising, since for fixed N, a discrete mixture with N components is at least as expressive as a continuous mixture approximated by N integration points, since the former has mixture components with free (private) parameters, while the latter has components which are determined via a shared neural network and thus entangled in a complex way. Moreover, the PCs of previous works have been deeper and used more sophisticated architectures than our continuous mixtures.

The main reason for the efficacy of our approach might be the continuity of the neural network, which topologically relates the latent and observable space, thus identifying some underlying latent structure—this is in fact one of the attractive and widely appreciated properties of VAEs. However, the effect of continuity on generalisation has not been much studied until the time of writing, and our results provide an interesting pointer in this regard.

Why does continuity promote generalisation, or act as some form of regularisation? For the one, there might be an Occam's razor effect at work, since our models are usually much smaller in terms of parameters, yet they are expressive due to the non-linear nature of neural nets. Furthermore, dependence among components introduced via the latent space might effectively facilitate learning, by avoiding redundant or 'dead' components, which have been observed in vanilla PCs (Dang et al. 2022).

These results have two important consequences for future work on tractable probabilistic models: (i) continuous latent spaces seem a valuable tool for learning tractable models and (ii) PCs in general seem to have untouched potential not yet exploited by existing learning methods.

8.6 Conclusion

In this chapter, we have investigated the marriage of continuous mixtures and tractable probabilistic models. We have observed that even when using simple structures and standard numerical integration techniques, continuous latent variables facilitate the learning of expressive PCs, as confirmed by SOTA results on many datasets. Moreover, we have proposed latent optimisation as an effective way to derive competitive mixture models even with relatively few components (integration points). We believe this is a promising research av-

enue both in hybrid inference (Tan and Peharz 2019) and learning of tractable probabilistic models in general.

Our model is not without limitations, however. In particular, numerical integration is a computationally expensive training approach—the cost of each gradient update is proportional to the number of integration points—and we assume fixed PC structures (independent of the latent variables) that have to be defined or learnt a-priori. Addressing these two issues are promising avenues for future work, especially with extensions to more complex structures, like HCLTs (Liu and Van den Broeck 2021) or EiNets (Peharz et al. 2020a).

Chapter 9

Conclusions and Future Work

This chapter concludes this thesis with first a summary of our work in Section 9.1, where we highlight our main findings and relate our results to the research questions outlined in Chapter 1. We also discuss the limitations of our work in Section 9.2 and propose some promising paths for future research in Section 9.3 that might address some of these limitations or simply answer some of the new questions that arose from our research.

9.1 Conclusion

Our work has provided some promising insights on learning (tractable) probabilistic models. These are motivated by the utility and need of *exact* and *efficient* probabilistic inference, be it in the context of learning algorithms, as in our work in Bayesian Networks, or tractable generative models, namely Probabilistic Circuits. We emphasise exact inference because it assures us of the accuracy and, to some extent, the reliability of our results. It is probably fair to say that exactness is often overlooked in the current machine learning literature, which most often favours the expressive power of the models. Of course, exact inference is only really useful if we can compute it in reasonable time with plausible resources. That is why the work in this thesis focus heavily on efficiency and tractability.

We started this journey with Bayesian Networks in Chapters 3 and 4. In this case the focus was not in tractable models per se—Bayesian Networks are not necessarily tractable themselves—but on the efficiency of exact learning algorithms. We then moved on to Probabilistic Circuits contributing new algo-

rithms for structure learning in Chapter 6 and parameter learning in Chapter 8 that extended the range of applications for PCs as well as uncovered some new avenues to increase the expressive power of these tractable models. We also studied ways to quantify the robustness of PCs in Chapter 7, drawing a few parallels with the model's likelihood function and accuracy. In the following, we dive a bit deeper into the conclusions we can derive from each of these chapters in relation to the research questions that we proposed in Chapter 1.

(Q1) **Can we still improve the computational efficiency of exact structure learning of BNs via pruning techniques?**

In Chapter 3, we have showed that by studying the properties of the gamma and likelihood functions we could derive new and tighter upper bounds for the Bayesian Dirichlet equivalent uniform (BDeu) score. That is promising since these new bounds allow us to prune the search space in exact structure learning more aggressively, especially when considering large or no maximum in-degree (maximum number of parents per variable). However, in this work we have not implemented these upper bounds into exact structure learning solvers, so we cannot answer for certain what the real impact of our bounds will be. This research also lead to a somewhat tangential result discussed in the rather short Chapter 4. We empirically showed that the BDeu score is highly sensitive to the prior associated with the Equivalent Sample Size or ESS, drawing attention to the fact that one needs to be careful when setting this hyperparameter since no amount of data seems to render the final learnt architecture fully independent of the ESS value.

(Q2) **How to improve the performance of PCs in classification tasks?**

Probabilistic Circuits are generative models by nature and often lag behind discriminative models, like Random Forests (Breiman 2001) and XGBoost (Chen and Guestrin 2016), in terms of classification accuracy. Nonetheless, our research has shown that we can significantly improve the performance of PCs in classification tasks via architecture design. In Chapter 7 we show that class-selective architectures, which simply dedicate the last layer to the class variable, already improve accuracy significantly in comparison to standard architectures that treat explanatory and target variables equally. Moreover, in the research discussed in Chapter 6 we showed that PCs generalise Decision Trees and that, consequently, we can extend any classification Decision Tree or Random Forest to a fully generative model represented by a PC. That gave rise to a new class of PCs,

named Generative Forests, that effectively borrows its learning algorithm from the literature on Decision Trees and by design achieves the exact same performance as Random Forests in classification tasks. These results set forth that the introduction of inductive biases in the PC architecture is a promising and already highly effective answer to this research question.

(Q3) **Can we leverage continuous latent variables in PCs?**

In Chapter 8 we proposed continuous mixtures of PCs as a way to introduce a single continuous latent variable into the realm of Probabilistic Circuits. In practice, we used a learnable function, a neural network, to map the latent space to the parameters of a PC with fixed structure. We showed that we can train and run inference on these models using out-of-the-shelf numerical integration methods with surprisingly good results. Even with simplistic PC architectures, namely fully factorised or Chow-Liu-Tree structures, our continuous mixtures outperformed state-of-the-art PCs in a number of density estimation tasks. With that we can confidently answer this research question with a resounding yes.

(Q4) **How to quantify the robustness or reliability of PCs?**

We have tried to answer this question by studying two different signals: ϵ-robustness (Maua et al. 2018) and the likelihood function. In our work, presented in Chapter 7, we investigated how to make ϵ-robustness more efficient via a new architecture, called class-selective, for which we can compute such robustness values with a single pass through the network. Moreover, we showed that ϵ-robustness is correlated with accuracy both in class-selective PCs and GeFs, which imbues ϵ-robustness with a notion of reliability. We also studied the likelihood function, which can be evaluated exactly in PCs and is also related to reliability as a means for outlier detection. Intuitively, a model is not expected to be reliable on out-of-domain samples since those are, by definition, statistically different from the training data. We showed that GeFs effectively identify outliers simply by monitoring the likelihood, which adds virtually no computational overhead on top of the regular inference routing for classification. This is somewhat surprising since deep generative models tend to be unreliable outlier detectors (Nalisnick et al. 2018), although some positive results had already been reported for PCs (Peharz et al. 2020b). Finally, we also contrasted ϵ-robustness and the likelihood function, showing they are quite distinct and capture different notions of uncertainty, both of which might be useful to quantify the reliability of a prediction.

9.2 Limitations

In this section, we discuss the limitations of our work. We outline what we believe to be the main drawbacks in our work on upper bounds for the BDeu score, and in the new Probabilistic Circuits models we proposed, namely Generative Forests and Continuous Mixtures.

9.2.1 BDeu Upper Bounds

Our new upper bounds are tighter and prune the search space more aggressively than previous results found in the literature, e.g. (de Campos and Ji 2011). However, a limitation of our results is that we cannot yet ascertain whether our theoretical complexity results match practice. It remains to be seen whether our upper bounds indeed improve the efficiency of exact structure learning algorithms when integrated into existing software, which we leave for future work.

9.2.2 Generative Forests

The limitations of Generative Forests (or GeFs) that we are going to discuss are in relation to Random Forests. That is because GeFs subsume Random Forests and could replace them in a number of applications where one needs to handle missing data or detect outliers besides good classification performance.

The first limitation we address is the computational overhead in comparison to traditional Random Forests, which unfortunately is non-negligible both in terms of memory requirements and inference cost. The memory overhead is obvious when we realise GeFs store, for each leaf of each tree, the parameters of a full joint distribution, which grows at best linearly in the number of variables. The additional inference cost in GeFs is not significant for complete data, as shown in Section 6.6, but might be much higher for incomplete data, since we might be forced to visit multiple branches of each tree; each split on a non-observed variable requires running both descendant branches. Of course, this additional computational and memory toll pales in comparison to the cost and size of modern machine learning models that have grown to billions, if not trillions, of parameters. Still, they might be significant in comparison to the very low computational demands of Random Forests; one of the main selling points of these models that makes them appealing to applications requiring low latency or disposing of little computing power.

One way we could address this limitation is to add a full joint distribution only to leaves with large numbers of nodes. The remaining leaves could simply

output a constant, since leaves with few training datapoints contribute less to the final result[1] and their density estimators are unlikely to be very reliable. That saves computational effort during training, as we do not require fitting a distribution on every leaf, and during inference since we do not have to evaluate the distribution of small leaves.

The second limitation relates to the additional modelling decisions and assumptions that GeFs impose. Notably, one has to decide on the form of the parametric distribution in each leaf, which requires extra information about the data and might introduce undesirable biases. That said, GeDTs (and GeFs) can maintain the same conditional distribution over the target variable of the underlying Random Forest, and thus these extra modelling decisions are innocuous to the classification function defined by the original model.

9.2.3 Continuous Mixtures of Tractable Probabilistic Models

Our empirical results show continuous latent variables are indeed useful tools for learning tractable probabilistic models or, to be more precise, Probabilistic Circuits. However, once again our models introduce additional computational cost. Training continuous mixtures via numerical integration is computationally expensive; the cost of each gradient step is directly proportional to the number of integration points used. Moreover, even though the models can be represented quite compactly via the neural network decoder, the full compiled PC can become quite large even for modest numbers of integration points.

All in all, we believe continuous latent variables or similarly motivated optimisation tools might be an important ingredient of future PC learning algorithms. That said, at this point these methods are limited to only one continuous latent variable and it is not entirely clear how to extend them to include multiple continuous latent variables and sophisticated latent structures in a single architecture. That could introduce a chain of numerical approximations that might become too imprecise or too expensive to run. For instance, consider two continuous latent variables $\mathbf{Z}_1$ and $\mathbf{Z}_2$ defining the following distribution

$$p(\mathbf{x}) = \int \int p(\mathbf{x}|\, \boldsymbol{\theta}(\mathbf{z}_2)) p(\mathbf{z}_2|\mathbf{z}_1) p(\mathbf{z}_1) d\mathbf{z}_2 d\mathbf{z}_1.$$

Each value that $\mathbf{Z}_1$ assumes defines a new distribution over $\mathbf{X}$ via $p(\mathbf{x}|\, \boldsymbol{\theta}(\mathbf{z}_2))$, where $\boldsymbol{\theta}$ is again some function mapping latent variables to PC parameters. If the numerical precision we need requires N integration points, then we have

[1] Sum-node weights are proportional to the number of training points in each of its children.

to sample N values from $p(\mathbf{z}_1)$ and N values *for each* $p(\mathbf{z}_2|\mathbf{z}_1)$. That means we need to evaluate $p(\mathbf{x}|\,\boldsymbol{\theta}(\mathbf{z}_2))$ a total of N^2 times, and the computational cost is exponential on the number of sequential continuous latent variables. A possible way to mitigate this exponential growth is through importance sampling, so that we can continue to use only N integration points at each step in the chain as in (Elvira et al. 2020), but we leave that for future work.

9.3 Future Research

In this section, we identify a few promising directions for future research, specifically in what touches Generative Forests and Continuous Mixtures.

9.3.1 Generative Forests

Our work presented in Chapter 6 was focused on classification tasks. The most natural next step for this research would be to extend Generative Forests to regression tasks as well. The challenge, as already discussed in Section 6.7, is that GeFs actually propagate distributions through the graph. For classification tasks the distributions are categorical, and thus it is straightforward to aggregate two distributions coming from different leaves; we simply compute a weighted average which returns another categorical distribution. That is not as simple when the target variable is continuous. Even if the leaf distributions are as simple as normal distributions, once we aggregate two normal distributions, we get a mixture that we can no longer represent with only two parameters as a normal distribution. That means that if the observed variables match multiples leaves, which happens in the case of missing data, we need to propagate increasingly complex distributions up through the network. This is a computational and coding challenge rather than a theoretical impediment, but it means Generative Forest would be more costly to run on regression than on classification tasks.

Another interesting research direction is to consider other tree learning algorithms, like gradient boosting (Friedman 2001), to learn the structure of GeFs. Unfortunately, this is probably only uncomplicated for binary classification. For regression problems we stumble on the challenges mentioned above and, for multi-class classification tasks, boosting methods often use one tree per class (Chen et al. 2016), which might be hard to represent with a compact GeF.

9.3.2 Continuous Mixtures

There are a few promising research avenues in expanding the use of continuous mixtures to other PC architectures and learning algorithms. The first one, that has already been mentioned when discussed limitations, is including multiple continuous latent variables into a single PC architecture. Another interesting path for future work is investigating how beneficial are continuous mixtures of large and sophisticated architectures, like Einets (Peharz et al. 2020a). For instance, the simple architectures we considered in Chapter 8 had no product nodes, which might change the learning dynamics significantly. Finally, perhaps the most promising and challenging research direction is making the structure also dependent on the continuous latent variable. One could consider differentiable DAG structure learners (Zheng et al. 2018) as a way to fit structure and parameters at the same time with gradient descent.

One could also investigate alternatives to numerical integration for learning continuous mixtures. As discussed in Section 8.4.1, we hypothesise that, with sufficient regularisation, other more efficient methods like variational training can yield continuous mixtures that are amenable to numerical integration, and hence can be effectively compiled to PCs. This could make continuous mixtures much more scalable, opening the way for PCs of comparable size to modern deep neural networks with billions of parameters.

Bibliography

T. Adel and C. De Campos. Learning bayesian networks with incomplete data by augmentation. In *Thirty-first aaai conference on artificial intelligence*, 2017. (Cited on page 11.)

H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998. (Cited on page 18.)

S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998. (Cited on pages 67 and 131.)

M. R. Amer and S. Todorovic. Sum product networks for activity recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(4):800–813, 2016. (Cited on page 92.)

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 214–223. PMLR, 2017. (Cited on page 123.)

S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. (Cited on page 45.)

M. Barlett and J. Cussens. Advances in bayesian network learning using integer programming. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 182–191, 2013. (Cited on page 49.)

M. Bartlett and J. Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017. (Cited on page 25.)

I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pages 247–256. Springer-Verlag, 1989. (Cited on pages 49, 50, and 52.)

J. Bekker, J. Davis, A. Choi, A. Darwiche, and G. Van den Broeck. Tractable learning for complex probability queries. *Advances in Neural Information Processing Systems*, 28, 2015. (Cited on pages 119 and 175.)

M. Benjumeda, P. Larrañaga, and C. Bielza. Learning bayesian networks with low inference complexity. *Progress in Artificial Intelligence*, 5(1):15–26, 2016. (Cited on page 45.)

M. Benjumeda, C. Bielza, and P. Larranaga. Learning tractable bayesian networks in the space of elimination orders. *Artificial Intelligence*, 274:66–90, 2019. (Cited on page 45.)

G. Biau, L. Devroye, and G. Lugosi. Consistency of Random Forests and Other Averaging Classifiers. *Journal of Machine Learning Research*, 9:2015–2033, 2008. (Cited on pages 72 and 84.)

P. Bojanowski, A. Joulin, D. Lopez-Pas, and A. Szlam. Optimizing the latent space of generative networks. In *Proceedings on the International Conference on Machine Learning (ICML)*, pages 600–609. PMLR, 2018. (Cited on pages xiv and 119.)

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012. (Cited on page 67.)

R. R. Bouckaert. *Bayesian belief networks: from construction to inference*. PhD thesis, Universiteit Utrecht, 1995. (Cited on page 17.)

J. Bradshaw, A. G. d. G. Matthews, and Z. Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv:1707.02476*, 2017. (Cited on page 100.)

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. (Cited on pages xiv, 71, 74, 84, and 142.)

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984. (Cited on pages xiii, 72, 74, 83, 87, and 172.)

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020. (Cited on page 1.)

A. Buchholz, F. Wenzel, and S. Mandt. Quasi-monte carlo variational inference. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 668–677. PMLR, 2018. (Cited on pages 117 and 128.)

H.-J. Bungartz and M. Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004. (Cited on page 115.)

W. Buntine. Theory refinement on bayesian networks. In *Uncertainty proceedings 1991*, pages 52–60. Elsevier, 1991. (Cited on pages 9, 16, 24, and 48.)

Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015. (Cited on page 117.)

S. E. Buttrey and C. Karo. Using k-nearest-neighbor classification in the leaves of a tree. *Computational Statistics & Data Analysis*, 40(1):27–37, 2002. (Cited on page 75.)

R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7: 1–49, 1998. (Cited on page 116.)

A. Cano, A. R. Masegosa, and S. Moral. A method for integrating expert knowledge when learning bayesian networks from data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(5):1382–1394, 2011. (Cited on page 9.)

E. Y.-J. Chen, Y. Shen, A. Choi, and A. Darwiche. Learning Bayesian networks with ancestral constraints. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 2325–2333. Curran Associates, Inc., 2016. (Cited on pages 25, 92, and 146.)

T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, volume 19(6), pages 785–794, 2016. (Cited on pages 101 and 142.)

W. C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai. Language modeling with sum-product networks. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 2098–2102, 2014. (Cited on page 92.)

D. M. Chickering. A transformational characterization of equivalent bayesian network structures. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 87–98, 1995. (Cited on pages 12 and 13.)

D. M. Chickering. Learning bayesian networks is np-complete. In *Learning from data*, pages 121–130. Springer, 1996. (Cited on page 12.)

D. M. Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002. (Cited on pages 13 and 14.)

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of bayesian networks is np-hard. *The Journal of Machine Learning Research*, 5: 1287–1330, 2004. (Cited on pages 12 and 25.)

S.-C. T. Choi, F. J. Hickernell, M. McCourt, and A. Sorokin. QMCPy: A quasi-Monte Carlo Python library, 2020+. URL `https://github.com/QMCSoftware/QMCSoftware`. (Cited on page 119.)

Y. Choi, A. Vergari, and G. Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models, 2020. URL `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`. (Cited on pages 55 and 68.)

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968. (Cited on pages xiii, 5, 19, 20, 55, and 119.)

D. Colombo, M. H. Maathuis, et al. Order-independent constraint-based causal structure learning. *J. Mach. Learn. Res.*, 15(1):3741–3782, 2014. (Cited on page 11.)

G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9:309–347, 1992. (Cited on pages 15, 16, 24, and 48.)

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022. (Cited on page 20.)

A. H. C. Correia and C. de Campos. Towards scalable and robust sum-product networks. In *International Conference on Scalable Uncertainty Management*, pages 409–422. Springer, 2019. (Cited on pages 87, 89, 92, and 174.)

A. H. C. Correia, C. de Campos, and L. C. van der Gaag. An experimental study of prior dependence in bayesian network structure learning. In *International Symposium on Imprecise Probabilities: Theories and Applications*, pages 78–81. PMLR, 2019. (Cited on page 17.)

A. H. C. Correia, J. Cussens, and C. de Campos. On pruning for score-based bayesian network structure learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2709–2718. PMLR, 2020a. (Cited on pages 8 and 12.)

A. H. C. Correia, R. Peharz, and C. de Campos. Joints in random forests. *Advances in Neural Information Processing Systems*, 33, 2020b. (Cited on pages 55, 61, and 101.)

A. H. C. Correia, G. Gala, E. Quaeghebeur, C. de Campos, and R. Peharz. Continuous mixtures of tractable probabilistic models. *arXiv preprint arXiv:2209.10584*, 2022. (Cited on page 60.)

P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, nov 2009. (Cited on page 100.)

J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 153–160. AUAI Press, 2011. (Cited on pages 12 and 25.)

J. Cussens. An upper bound for BDeu local scores. Proc. ECAI-2012 workshop on algorithmic issues for inference in graphical models (AIGM), 2012. (Cited on page 29.)

J. Cussens and M. Bartlett. GOBNILP 1.6.2 User/Developer Manual, 2015. (Cited on pages 30 and 42.)

J. Cussens, M. Bartlett, E. M. Jones, and N. A. Sheehan. Maximum likelihood pedigree reconstruction using integer linear programming. *Genetic Epidemiology*, 37(1):69–83, 2013. (Cited on page 8.)

B. Dai and D. Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=B1e0X3C9tQ`. (Cited on page 131.)

M. Dang, A. Vergari, and G. Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *International Conference on Probabilistic Graphical Models*, pages 137–148. PMLR, 2020. (Cited on page 179.)

M. Dang, A. Liu, and G. Van den Broeck. Sparse probabilistic circuits via pruning and growing. In *The 5th Workshop on Tractable Probabilistic Modeling*, 2022. (Cited on page 138.)

A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003. (Cited on pages 55, 57, 59, 66, 67, and 96.)

C. de Campos and Q. Ji. Properties of Bayesian Dirichlet scores to learn Bayesian network structures. In *Conference on Advancements in Artificial Intelligence (AAAI)*, pages 431–436, 2010. (Cited on page 29.)

C. de Campos and Q. Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011. (Cited on pages 2, 25, 29, 30, 42, 49, and 144.)

C. de Campos, Z. Zeng, and Q. Ji. Structure learning of Bayesian networks using constraints. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 382, pages 113–120. ACM, 2009. (Cited on pages 12 and 25.)

L. M. de Campos and J. G. Castellano. Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning*, 45 (2):233–254, 2007. (Cited on page 9.)

R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2-3):73–106, 2007. (Cited on page 64.)

O. Delalleau and Y. Bengio. Shallow vs. Deep Sum-Product Networks. In *Advances in Neural Information Processing Systems*, pages 666—-674. Curran Associates, Inc., 2011. (Cited on page 92.)

A. Dennis and D. Ventura. Greedy structure search for sum-product networks. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. (Cited on page 64.)

L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, New York, 1996. (Cited on pages 80 and 83.)

N. Di Mauro, A. Vergari, T. M. Basile, and F. Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 203–219, 2017. (Cited on page 75.)

D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. (Cited on pages 41, 42, 49, 53, 101, 104, and 105.)

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12 (7), 2011. (Cited on page 66.)

G. Elidan and S. Gould. Learning bounded treewidth bayesian networks. *Advances in Neural Information Processing Systems*, 21, 2008. (Cited on page 45.)

V. Elvira, L. Martino, and P. Closas. Importance gaussian quadrature. *IEEE Transactions on Signal Processing*, 69:474–488, 2020. (Cited on page 146.)

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014. (Cited on page 70.)

E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten. Using model trees for classification. *Machine learning*, 32(1):63–76, 1998. (Cited on page 75.)

J. H. Friedman. A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, C-26(4):404–408, 1977. (Cited on pages 72, 87, and 172.)

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. (Cited on page 146.)

N. Friedman. The bayesian structural em algorithm. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 129–138, 1998. (Cited on page 11.)

J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004. (Cited on page 75.)

D. Geiger, D. Heckerman, H. King, and C. Meek. Stratified exponential families: graphical models and model selection. *The Annals of statistics*, 29(2):505–529, 2001. (Cited on page 18.)

R. Gens and P. Domingos. Learning the Structure of Sum-Product Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 28, pages 229–264, 2013. (Cited on pages 59, 61, 63, 66, 78, 80, 87, 101, 102, 120, 174, and 179.)

T. Gerstner and M. Griebel. Sparse grids. In R. Cont, editor, *Encyclopedia of Quantitative Finance*. John Wiley and Sons, 2010. (Cited on page 116.)

Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015. (Cited on page 110.)

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. (Cited on page 57.)

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. (Cited on pages xiv, 1, 110, and 117.)

L. Gordon and R. A. Olshen. Asymptotically efficient solutions to the classification problem. *The Annals of Statistics*, pages 515–533, 1978. (Cited on page 72.)

A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 203–211. SIAM, 2003. (Cited on page 75.)

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *Advances in Neural Information Processing Systems*, 30, 2017. (Cited on page 123.)

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. (Cited on page 74.)

D. M. Haughton. On the choice of a model to fit data from an exponential family. *The annals of statistics*, pages 342–355, 1988. (Cited on page 18.)

D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3): 197–243, 1995. (Cited on pages 9, 15, 16, 24, 48, and 51.)

D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016. (Cited on page 101.)

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. (Cited on page 1.)

Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings Of 1st Pacific-Asia Conference on Knowledge Discovery And Data Mining*, 1997. (Cited on page 173.)

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 448–456. PMLR, 2015. (Cited on pages 120 and 175.)

T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 358–365, 2010. Journal of Machine Learning Research Workshop and Conference Proceedings. (Cited on page 25.)

E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003. (Cited on pages 2 and 110.)

Kaggle. Kaggle's State of Data Science and Machine Learning 2019. Technical report, Kaggle, 2019. URL `https://www.kaggle.com/kaggle-survey-2019`. (Cited on page 70.)

M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. (Cited on page 62.)

P. Khosravi, Y. Liang, Y. Choi, and G. Van Den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2716–2724. AAAI Press, 2019. (Cited on page 75.)

P. Khosravi, A. Vergari, Y. Choi, Y. Liang, and G. V. d. Broeck. Handling missing data in decision trees: A probabilistic approach. *arXiv:2006.16341*, 2020. (Cited on page 75.)

H. Kim and W.-Y. Loh. Classification trees with bivariate linear discriminant node models. *Journal of Computational and Graphical Statistics*, 12(3):512–530, 2003. (Cited on page 75.)

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on pages 66, 120, and 176.)

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014. arXiv:1312.6114. (Cited on pages xiv, 1, 59, 78, 110, 113, 114, 117, 122, 123, and 175.)

D. Kisa, G. V. den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *Knowledge Representation and Reasoning Conference*, 2014. (Cited on page 55.)

R. Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In *Knowledge Discovery and Data Mining (KDD)*, pages 202–2017, 1996. (Cited on page 75.)

M. Koivisto. Parent Assignment Is Hard for the MDL, AIC, and NML Costs. In *Computational Learning Theory (COLT)*, volume 4005, pages 289–303. Springer, 2006. (Cited on page 25.)

M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004. (Cited on page 25.)

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. (Cited on pages 8, 13, 19, and 21.)

B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2012. (Cited on page 95.)

W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952. (Cited on page 62.)

S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. (Cited on page 20.)

J. Kwisthout, H. L. Bodlaender, and L. C. van der Gaag. The necessity of bounded treewidth for efficient inference in bayesian networks. In *ECAI*, volume 215, pages 237–242, 2010. (Cited on page 45.)

N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine learning*, 59 (1-2):161–205, 2005. (Cited on page 75.)

H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011. (Cited on pages 119, 125, and 175.)

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on pages 119 and 175.)

Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database, 2010. (Cited on page 105.)

P. l'Ecuyer. Randomized quasi-monte carlo: An introduction for practitioners. In *International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 29–52. Springer, 2016. (Cited on pages 116, 128, and 176.)

B. Levin and J. Reeds. Compound multinomial likelihood functions are unimodal: Proof of a conjecture of I. J. Good. *The Annals of Statistics*, 5(1):79–87, 1977. (Cited on page 37.)

Y. Liang, J. Bekker, and G. Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017. (Cited on page 179.)

R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019. (Cited on page 80.)

A. Liu and G. Van den Broeck. Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34, 2021. (Cited on pages 21, 67, 120, 139, and 179.)

F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008. (Cited on page 101.)

W.-Y. Loh. Improving the precision of classification trees. *The Annals of Applied Statistics*, pages 1710–1737, 2009. (Cited on page 75.)

G. Louppe. *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liege, 2014. (Cited on page 84.)

D. Lowd and J. Davis. Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010. (Cited on pages 119 and 175.)

G. Lugosi, A. Nobel, et al. Consistency of data-driven histogram methods for density estimation and classification. *The Annals of Statistics*, 24(2):687–706, 1996. (Cited on pages 81, 82, and 83.)

D. J. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80, 1995. (Cited on pages 113 and 117.)

D. Madigan, S. A. Andersson, M. D. Perlman, and C. T. Volinsky. Bayesian model averaging and model selection for markov equivalence classes of acyclic digraphs. *Communications in Statistics–Theory and Methods*, 25(11):2493–2519, 1996. (Cited on page 13.)

J. Martens and V. Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014. (Cited on page 68.)

D. D. Maua, D. Conaty, F. G. Cozman, K. Poppenhaeger, and C. de Campos. Robustifying Sum-Product Networks. *International Journal of Approximate Reasoning*, 101:163 – 180, 2018. (Cited on pages 3, 94, 96, and 143.)

D. D. Mauá, F. G. Cozman, D. Conaty, and C. P. Campos. Credal sum-product networks. In A. Antonucci, G. Corani, I. Couso, and S. Destercke, editors, *Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications*, volume 62 of *Proceedings of Machine Learning Research*, pages 205–216. PMLR, 10–14 Jul 2017. (Cited on pages 3, 5, 92, and 93.)

J. H. McDonald. *Handbook of biological statistics*, volume 2. sparky house publishing Baltimore, MD, 2009. (Cited on page 62.)

J. Mingers. Expert systems—rule induction with statistical data. *Journal of the operational research society*, 38(1):39–47, 1987. (Cited on page 74.)

A. Mnih and D. Rezende. Variational inference for monte carlo objectives. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2188–2196, 2016. (Cited on page 117.)

A. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998. (Cited on page 44.)

S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In *Proceedings of European Simulation and Modelling Conference-ESM'2011*, pages 117–121. EUROSIS-ETI, 2011. (Cited on page 88.)

E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations (ICLR)*, 2018. (Cited on pages 98 and 143.)

A. Nath and P. Domingos. Learning tractable probabilistic models for fault localization. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 1294–1301, 2016. (Cited on page 92.)

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. (Cited on pages xiv, 119, and 175.)

S. Nie, D. D. Mauá, C. P. De Campos, and Q. Ji. Advances in learning bayesian networks of bounded treewidth. *Advances in Neural Information Processing Systems*, 27, 2014. (Cited on page 45.)

S. Nie, C. P. de Campos, and Q. Ji. Efficient learning of bayesian networks with bounded tree-width. *International Journal of Approximate Reasoning*, 80: 412–427, 2017. (Cited on page 45.)

G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.*, 22(57):1–64, 2021. (Cited on pages 1 and 98.)

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. (Cited on page 119.)

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. (Cited on page 119.)

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988. (Cited on pages 8, 24, and 64.)

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014. (Cited on page 8.)

K. Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. (Cited on page 62.)

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (Cited on pages 101 and 173.)

R. Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Graz University of Technology, 2015. (Cited on page 111.)

R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. *Proceedings of the International Conference on Machine Learning (ICML)*, 32, 2014. (Cited on pages 77 and 96.)

R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752, 2015. (Cited on pages 57, 58, 60, 93, and 136.)

R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016. (Cited on pages 59, 60, 66, 78, 111, and 131.)

R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the Interna-*

tional Conference on Machine Learning (ICML), pages 7563–7574. PMLR, 2020a. (Cited on pages xiii, 61, 66, 111, 120, 128, 130, 131, 139, 147, 176, and 179.)

R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b. (Cited on pages 61, 98, 120, 143, and 179.)

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 337–346, 2011. (Cited on pages 55, 57, 66, 67, 68, 78, 92, 128, and 176.)

P. Probst and A. L. Boulesteix. To tune or not to tune the number of trees in random forest. *Journal of Machine Learning Research*, 18:1–8, 2018. (Cited on page 87.)

A. Pronobis and R. P. Rao. Learning deep generative spatial models for mobile robots. *IEEE International Conference on Intelligent Robots and Systems*, pages 755–762, 2017. (Cited on page 92.)

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. (Cited on pages 71 and 74.)

J. R. Quinlan. Decision trees as probabilistic classifiers. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 31–37. Elsevier, 1987a. (Cited on pages 72, 87, and 172.)

J. R. Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987b. (Cited on page 74.)

J. R. Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. World Scientific, 1992. (Cited on page 75.)

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. (Cited on pages 128 and 175.)

T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645. Springer, 2014. (Cited on pages xiii, 21, 55, 61, 64, 65, 66, 71, 75, and 77.)

P. Ram and A. G. Gray. Density estimation trees. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 627–635, 2011. (Cited on page 75.)

A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8821–8831. PMLR, 2021. (Cited on page 1.)

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1530–1538, 2015. (Cited on pages 78, 110, 113, and 114.)

T. Richardson. A characterization of markov equivalence for directed cyclic graphs. *International Journal of Approximate Reasoning*, 17(2-3):107–162, 1997. (Cited on page 13.)

R. W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial mathematics V*, pages 28–43. Springer, 1977. (Cited on page 12.)

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 710–718. PMLR, 2014. (Cited on pages 120 and 179.)

S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345, 1999. (Cited on page 68.)

M. Saar-Tsechansky and F. Provost. Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8:1625–1657, 2007. (Cited on page 173.)

K. Sachs, O. Perez, D. Pe, D. Lauffenburger, and G. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308 (5721):523–529, 2005. (Cited on pages 49, 50, and 52.)

M.-a. Sato. Fast learning of on-line em algorithm. *Rapport Technique, ATR Human Information Processing Research Laboratories*, 1999. (Cited on pages 67 and 131.)

M. Scanagatta, C. P. de Campos, G. Corani, and M. Zaffalon. Learning bayesian networks with thousands of variables. *Advances in Neural Information Processing Systems*, 28, 2015. (Cited on page 45.)

M. Scanagatta, G. Corani, C. P. De Campos, and M. Zaffalon. Learning treewidth-bounded bayesian networks with thousands of variables. *Advances in Neural Information Processing Systems*, 29, 2016. (Cited on page 45.)

M. Scanagatta, G. Corani, M. Zaffalon, J. Yoo, and U. Kang. Efficient learning of bounded-treewidth bayesian networks from complete and incomplete data sets. *International Journal of Approximate Reasoning*, 95:152–166, 2018. (Cited on page 45.)

M. Scutari. An Empirical-Bayes Score for Discrete Bayesian Networks. *Journal of Machine Learning Research (Proceedings Track, PGM 2016)*, 52:438–448, 2016. (Cited on pages 49 and 52.)

M. Scutari. Dirichlet bayesian network scores and the maximum relative entropy principle. *Behaviormetrika*, 45(2):337–362, Oct 2018. (Cited on pages 18, 49, and 52.)

B. M. Sguerra and F. G. Cozman. Image Classification Using Sum-Product Networks for Autonomous Flight of Micro Aerial Vehicles. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 139–144, 2016. (Cited on page 92.)

X. Shao, A. Molina, A. Vergari, K. Stelzner, R. Peharz, T. Liebig, and K. Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pages 401–412. PMLR, 2020. (Cited on page 114.)

A. Shih, D. Sadigh, and S. Ermon. Hyperspns: Compact and expressive probabilistic circuits. *Advances in Neural Information Processing Systems*, 34, 2021. (Cited on page 114.)

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal bayesian network structure. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 445–452, 2006. (Cited on pages 49 and 50.)

T. Silander, P. Kontkanen, and P. Myllymäki. On sensitivity of the map bayesian network structure to the equivalent sample size parameter. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 360–367, 2007. (Cited on pages 17 and 49.)

T. Silander, T. Roos, P. Kontkanen, and P. Myllymäki. Factorized normalized maximum likelihood criterion for learning bayesian network structures. In *Proceedings of the 4th European workshop on probabilistic graphical models (PGM-08)*, pages 257–272, 2008. (Cited on pages 9 and 17.)

N. Slobodianik, D. Zaporozhets, and N. Madras. Strong limit theorems for the bayesian scoring criterion in bayesian networks. *Journal of Machine Learning Research*, 10(7), 2009. (Cited on page 18.)

S. A. Smolyak. Interpolation and quadrature formulas for the classes W_s^α and E_s^α. In *Doklady Akademii Nauk*, volume 131, pages 1028–1031. Russian Academy of Sciences, 1960. (Cited on page 115.)

P. Smyth, A. Gray, and U. M. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Machine Learning Proceedings 1995*, pages 506–514. Elsevier, 1995. (Cited on page 75.)

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2256–2265. PMLR, 2015. (Cited on page 1.)

D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell. Bayesian Analysis in Expert Systems. *Statistical Science*, 8(3):219–247, 1993. (Cited on pages 49, 50, and 52.)

P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman. *Causation, prediction, and search*. MIT press, 2000. (Cited on page 11.)

H. Steck. Learning the bayesian network structure: dirichlet prior versus data. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 511–518, 2008. (Cited on pages 49 and 50.)

H. Steck and T. S. Jaakkola. On the Dirichlet Prior and Bayesian Regularization. *Advances in Neural Information Processing Systems*, pages 713–720, 2003. (Cited on pages 49, 50, and 51.)

D. J. Stekhoven and P. Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012. (Cited on pages 87 and 173.)

J. Suzuki. Learning bayesian belief networks based on the minimum description length principle: basic properties. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 82(10):2237–2245, 1999. (Cited on page 18.)

P. L. Tan and R. Peharz. Hierarchical decompositional mixtures of variational autoencoders. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 6115–6124. PMLR, 2019. (Cited on pages 59 and 139.)

M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Procedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 584–590, 2005. (Cited on page 27.)

L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. (Cited on pages 131, 132, and 176.)

T. M. Therneau, E. J. Atkinson, et al. An introduction to recursive partitioning using the rpart routines, 1997. (Cited on pages 72, 87, and 172.)

J. Tomczak and M. Welling. Vae with a vampprior. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1214–1223. PMLR, 2018. (Cited on page 111.)

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006. (Cited on page 51.)

M. Ueno. Learning networks determined by the ratio of prior and data. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 598–605, 2010. (Cited on pages 49, 50, 52, and 53.)

M. Ueno. Robust learning Bayesian networks for prior belief. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 698–707, 2011. (Cited on pages 49 and 50.)

B. Uria, I. Murray, and H. Larochelle. Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, 26, 2013. (Cited on page 131.)

A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30, 2017. (Cited on pages 128 and 175.)

J. Van Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012. (Cited on pages 119 and 175.)

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi:10.1145/2641190.2641198. URL `http://doi.acm.org/10.1145/2641190.2641198`. (Cited on pages 88 and 171.)

V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. (Cited on pages 81 and 82.)

A. Vergari, R. Peharz, N. Di Mauro, A. Molina, K. Kersting, and F. Esposito. Sum-product autoencoding: Encoding and decoding representations using sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. (Cited on page 60.)

A. Vergari, Y. Choi, R. Peharz, and G. Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. `http://starai.cs.ucla.edu/slides/AAAI20.pdf`, 2020. Tutorial at AAAI 2020. (Cited on pages 55, 57, 59, 68, 96, and 120.)

P. Walley. *Statistical Reasoning with Imprecise Probabilities*. Chapman & Hall, 1991. (Cited on page 5.)

J. Wang and G. Wang. Hierarchical Spatial Sum–Product Networks for Action Recognition in Still Images. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(1):90–100, 2018. (Cited on page 92.)

K. Weierstrass. Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885. (Cited on page 115.)

E. V. Wolputte, E. Korneva, and H. Blockeel. Mercs: Multi-directional ensembles of regression and classification trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. (Cited on page 75.)

K. Wu, K. Zhang, W. Fan, A. Edwards, and S. Y. Philip. Rs-forest: A rapid density estimator for streaming anomaly detection. In *2014 IEEE International Conference on Data Mining*, pages 600–609. IEEE, 2014. (Cited on page 75.)

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. (Cited on pages 105, 119, and 175.)

C. Yuan and B. Malone. An improved admissible heuristic for learning optimal Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 924–933, Catalina Island, CA, 2012. (Cited on pages 25 and 49.)

C. Yuan and B. Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, October 2013. (Cited on page 25.)

C. Yuan, B. Malone, and X. Wu. Learning optimal bayesian networks using a* search. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. (Cited on page 12.)

M. Zaffalon. The naive credal classifier. *Journal of statistical planning and inference*, 105(1):5–21, 2002. (Cited on page 97.)

M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (Cited on page 66.)

H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and bayesian networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 116–124. PMLR, 2015. (Cited on page 67.)

H. Zhao, P. Poupart, and G. Gordon. A unified approach for learning the parameters of sum-product networks. *arXiv preprint arXiv:1601.00318*, 2016. (Cited on page 111.)

X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. Dags with no tears: Continuous optimization for structure learning. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on pages 12 and 147.)

Appendix

This appendix contains detailed experimental description and extra results pertaining to Chapters 6 and 8.

A Chapter 6

All 21 datasets we used in the experiments were obtained directly from the OpenML-CC18 benchmark web-page[2] (Vanschoren et al. 2013). The only preprocessing step was standardising continuous features (mean $\mu = 0$ and standard deviation $\sigma = 1$) and mapping categorical features to $\{0, \ldots, K_i - 1\}$. The datasets as well as the source code are available at `https://github.com/alcorreia/gefs`. In the following we briefly discuss each of the methods and their implementations. The source code is all in Python 3 and all experiments were run in a single laptop with a modern CPU.

A.1 Random Forest implementation

In all experiments, the structure of all models is kept the same, that is, they are all derived from the same Random Forest and thus share the same partition of the feature space. For every dataset, the Random Forests were composed of 100 'deep' trees, that is, the only stop criterion is the impurity of the class variable, possibly leading to many leaves with a single sample. Each tree is learned on a bootstrap sample of the same size of the training dataset, and each split only evaluates $\sqrt{m}$ variables, with m the total number of features. We use the Gini impurity measure as the criterion to select the best split in the decision-tree learning and rank surrogate splits according to how well they predict the

[2] `https://www.openml.org/s/99/data`

best split, as in (Therneau et al. 1997). The trees are all binary, with splits on categorical variables defined by two subsets of the possible instantiations. That is somewhat different from other implementations, where the splits are either 'full', yielding one child per category, or given by a threshold, which implicitly assumes categorical variables are ordinal.

A.2 'Built-in' Methods

These are methods for treating missing values that do not require external models, and hence are 'built-in' into the decision tree structure. In fact, they consist of slight modifications to the inference procedure.

Surrogate splits (Breiman et al. 1984; Therneau et al. 1997). During training, once the best split is defined, one ranks alternative splits according to the number of instances that they send to the same branch as the best split. At test time, if the split variable is not observed, one tries the surrogate splits in order (starting with that which most resembles the best split). If none of the surrogate split variables is available, the instance is sent to the branch with the highest number of data points at training time. Surrogate splits have two notable drawbacks: (i) their performance is heavily dependent on the correlation between variables; (ii) they require storing every possible split to be guaranteed to work for all missing-value configurations, which is rather computational and memory intensive, especially for large ensembles.

Friedman method (Friedman 1977; Quinlan 1987a). Whenever a split variable is not observed, one follows both branches of the tree. That means any instance with missing value is mapped to multiple leaves, and the final prediction is given by the majority class across the sum of the counts of all these leaves. If $C^{\mathcal{A}}(j)$ gives the number of training instances of class j in cell $\mathcal{A}$, we can write Friedman's methods as

$$f(\mathbf{x}) = \underset{j \in \{1,\ldots,K\}}{\arg\max} \sum_{\mathcal{A} \in \boldsymbol{\mathcal{A}}} \mathbb{1}(\mathbf{x} \in \mathcal{A}) C^{\mathcal{A}}(j),$$

$$C^{\mathcal{A}}(j) = \sum_{i=1}^{n} \mathbb{1}(\mathbf{x}_i \in \mathcal{A}) \mathbb{1}(y_i = j),$$

where i runs through the n training instances $(\mathbf{x}_i, y_i)$, and j runs through the K possible classes. Note that Friedman's method can be seen as a simplified

version of GeFs where the density over explanatory variables is constant and the same in every leaf.

A.3 Imputation methods

It is not surprising that most of the work on handling missing data in decision trees and random forests rely on data imputation (Saar-Tsechansky and Provost 2007). That is, another or multiple other models are used to predict the missing values before feeding the data to the tree-based classifier. In the experiments we compare a few different types of imputation methods:

Mean Missing values are imputed with the mean for continuous variables or the most frequent observation for categorical variables.

KNN Similar to the simple method above but the means or most frequent values are taken over the K-nearest neighbours. We use a standard K-nearest neighbour implementation from `scikit-learn` (Pedregosa et al. 2011) with K=7. However, the distance function is updated to better accommodate mixed data types. Following, Huang et al. (Huang 1997), we define the distance measure as

$$d(\mathbf{x}_a, \mathbf{x}_b) = \gamma \sum_{i=0}^{m_0} w_i \delta(\mathbf{x}_a[i], \mathbf{x}_b[i]) + \sum_{i=m_0}^{m_1} w_i \sqrt{(\mathbf{x}_a[i] - \mathbf{x}_b[i])^2},$$

where γ is a parameter representing the relative importance of categorical and numerical features, w_i is the weight of feature i, and, without loss of generality, we assume features are ordered so that the first m_0 variables are categorical. The δ function is simply the Hamming distance: $\delta(\mathbf{x}_a[i], \mathbf{x}_b[i]) = 1$ if $\mathbf{x}_a[i] \neq \mathbf{x}_b[i]$, and $\delta(\mathbf{x}_a[i], \mathbf{x}_b[i]) = 0$ otherwise. As we have no reason to favour any feature or feature type, we set both γ and every w_i to one.

MissForest (Stekhoven and Bühlmann 2012). For each variable $X_i \in \mathcal{X}$, one learns a Random Forest (classifier/regressor) that is used to predict unobserved values of X_i given the other variables $\mathcal{X} \setminus X_i$. As more than one variable might be unobserved, MissForest starts by imputing missing values with the mean (or mode) and then iteratively updates its initial guess using the Random Forest predictors. The original MissForest algorithm proposed in (Stekhoven and Bühlmann 2012) also updates the Random Forest predictors at every iteration.

However, in our experiments that would allow MissForest to exploit test data information, which could compromise the results. Therefore, we fit the Random Forest predictors in the training data only and keep them fixed at test time. Note that the algorithm remains iterative, since the imputed values are still fed to the predictors in the next iteration. We use a standard Python implementation of MissForests from `missingpy`—adapted to accommodate the changes mentioned above—which relies on the `scikit-learn` implementation of Random Forests.

A.4 Generative Forests

Vanilla GeFs What we call *vanilla* GeF, or simply GeF, is a model where the distribution at the leaves is given by a fully factorised model, that is, for each leaf v, $p_v(\mathbf{x}, y) = p_v(x_1)p_v(x_2)\dots p_v(x_m)p_v(y)$. This is probably the simplest model that one can fit at the leaves and is clearly *class-factorised*. Therefore, vanilla GeFs preserve full backward-compatibility with the original RF, yielding the exact same prediction function for complete data.

GeF with LearnSPN For GeF(LearnSPN) and GeF$^+$(LearnSPN), the LearnSPN algorithm (Gens and Domingos 2013) is run only at leaves with more than 30 samples, and smaller leaves are modelled by a fully factorised model as in vanilla GeFs. That saves computational time with little performance impact, as the model derived from LearnSPN with few samples would be similarly simplistic. We run the LearnSPN algorithm as follows: sum nodes split the samples via K-means clustering with K=2, and product nodes split the variables with an independence threshold of 0.001 (pair of variables for which the independence test yields a p-value lower than the threshold are considered independent). We do not force independence between the class Y and input variables $\boldsymbol{X}$ in LearnSPN, which explains why, in contrast to GeF, GeF(LearnSPN) does not necessarily yield the same predictions as the original Random Forest.

LearnSPN Similarly, we also learn a PC by applying the LearnSPN algorithm (Gens and Domingos 2013) to the entire dataset. The hyperparameters for this experiment are the same as for GeFs with LearnSPN, but we use a variant of LearnSPN that yields class-selective PCs, which have been shown to outperform standard LearnSPN in classification tasks (Correia and de Campos 2019).

B Chapter 8

B.1 Model Description

Latent Space

We use latent variables of dimension $d{=}4$ for the 20 density estimation benchmarks (Lowd and Davis 2010; Van Haaren and Davis 2012; Bekker et al. 2015), and $d{=}16$ for the image datasets, Binary MNIST (Larochelle and Murray 2011), MNIST (LeCun et al. 1998), Fashion-MNIST (Xiao et al. 2017) and Street View House Numbers (SVHN) (Netzer et al. 2011). In all cases, the latent space was distributed as a standard normal, $\mathbf{z} \sim \mathcal{N}(0,1)^d$, and integration points $\{\mathbf{z}_i\}_{i=1}^N$ were generated via a low-discrepancy lattice sequence. In practice, quasi-Monte Carlo methods are designed for sequences $\{\mathbf{u}_i\}_{i=1}^N$ mimicking a uniform distribution $\mathcal{U}(0,1)^d$. However, for most commonly used distributions, it is easy to map uniform samples to the random variable of interest. For instance, for $\mathbf{Z}$ distributed as a multivariate normal with mean μ and covariance Σ, we can construct a sequence $\{\mathbf{z}_i\}_{i=1}^N$ from $\{\mathbf{u}_i\}_{i=1}^N$ as

$$\mathbf{z}_i = \Phi^{-1}(\mathbf{u}_i)\Sigma^{1/2} + \mu,$$

where Φ^{-1} is the inverse CDF of a standard normal distribution. We chose a standard normal prior to facilitate the comparison to VAEs (Kingma and Welling 2014). We have tried other distributions in preliminary experiments, especially $\mathcal{U}(0,1)^d$, but have not observed significant differences in performance.

Decoder architecture

In all experiments, only two architectures were considered depending on the type of data. For binary datasets, the decoder was a multi-layer perceptron (MLP) with 6 layers of progressively increasing hidden size (from latent dimension to input dimension). For non-binary image datasets we used a convolutional architecture similar to that of DCGAN (Radford et al. 2015) but we included residual convolutional blocks as in (Van Den Oord et al. 2017). The hyperparameters of these architectures were kept the same for all experiments, only varying input and output dimensions to accommodate different data dimensionality or parametrisation at the leaves (normal vs. categorical distributions). In all cases, we used LeakyReLU activations and batch normalisation (Ioffe and Szegedy 2015).

Structure in Tractable Probabilistic Models

Our method consists of a continuous mixture of tractable probabilistic models. The parameters of these models are a function of the continuous latent variables, and thus learnt, but their structure is assumed and has to be defined a priori. As discussed in the main text, we consider two types of structure: a fully factorised distribution $\mathcal{G}_{\mathsf{F}}$ and a Chow-Liu Tree $\mathcal{G}_{\mathsf{CLT}}$, which is learnt a priori using the training data. Ultimately, both structures are a composition of univariate distributions. For all these models, we use Bernoulli distributions with learnable parameters for binary data, and 256-dimensional categorical distributions for non-binary image data. We also considered treating images as continuous data, in which case we use normal distributions at the leaves and normalise the images to $[0, 1]$ as is common in the literature (Theis et al. 2015): we added uniform noise to each image and divided pixel values by 256. See Appendix E for results in this setting.

In the case of Einets (Peharz et al. 2020a), we used the publicly available implementation by Peharz et al.[3] with the exact same leaf distributions as just described above, and trained via EM. We used Poon-Domingos architectures (Poon and Domingos 2011) of different sizes: one partitioning the image space into 4×4 contiguous square blocks and another partitioning the image space into 7×7 blocks in the case of MNIST and into 8×8 blocks in the case of SVHN.

Training

We train via numerical integration with Randomised quasi-Monte Carlo (or RQMC). At each training step, we generate a new low-discrepancy lattice sequence via *random shifting*. That is, we generate a lattice sequence $\{\mathbf{u}_i\}_{i=1}^N$ with each $\mathbf{u}_i$ in $(0, 1)^d$ and, at each iteration, we shift it by adding a single random point to all other points in the sequence modulo 1.

$$\mathbf{u}_i' = \mathbf{u}_i + \mathbf{u}_{shift} (\mathrm{mod} 1), \;\; \mathbf{u}_{shift} \sim \mathcal{U}(0, 1)^d.$$

After shifting, $\{\mathbf{u}_i\}_{i=1}^N$ is mapped to $\{\mathbf{z}_i\}_{i=1}^N$ via a suitable transformation as described above. This is the simplest form of randomisation in RQMC and works well in practice. We recommend (l'Ecuyer 2016) for a good overview of RQMC methods. We train all models (including $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$) for 300 epochs using Adam (Kingma and Ba 2014) with learning rate of $1e^{-3}$, β_1=0.9 and β_2=0.999. We also track performance on the validation set and use early-stopping with a patience of 15 to avoid overfitting.

[3] https://github.com/cambridge-mlg/EinsumNetworks.

In the experiments with binary datasets, we used a batch size of 128 and 2^{10} integration points to train both $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$. We report results across 5 random seeds, which we set to $\{0, 1, 2, 3, 4\}$. At test time, RQMC sequences are still stochastic, so we evaluate models with random seed set to 42 for reproducibility. In the experiments with image datasets, we used a larger batch size of 512 and also increased the number of integration points 2^{14}.

B.2 Additional Experimental Results

In this subsection, we present additional results on continuous mixtures, showing test log-likelihoods on the 20 binary density estimation benchmarks for different numbers of integration points in Tables B.1 and B.2. We also include latent optimisation results for $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ in Table B.3, showing significant improvements especially for few integration points. The same trend can be observed in Figure B.1, where we show the performance of $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$, $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$, $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ for all 20 density estimation benchmarks for different numbers of integration points.

Table B.1: Average $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ test log-likelihoods across 5 random seeds on 20 density estimation benchmarks. Higher is better

Dataset	Mix-2^7	Mix-2^8	Mix-2^9	Mix-2^{10}	Mix-2^{11}	Mix-2^{12}	Mix-2^{13}
accidents	-38.78±0.22	-36.62±0.17	-34.85±0.11	-33.94±0.05	-33.58±0.05	-33.41±0.05	-33.27±0.03
ad	-42.89±2.39	-32.50±1.44	-23.79±0.93	-20.42±0.14	-19.59±0.11	-19.18±0.10	-18.71±0.15
baudio	-40.24±0.07	-39.76±0.03	-39.34±0.01	-39.14±0.01	-39.07±0.01	-39.05±0.01	-39.02±0.01
bbc	-249.58±0.66	-245.66±0.68	-243.06±0.34	-241.54±0.31	-240.78±0.36	-240.53±0.32	-240.19±0.29
bnetflix	-57.35±0.09	-56.65±0.12	-56.02±0.01	-55.71±0.01	-55.58±0.01	-55.54±0.01	-55.49±0.02
book	-34.51±0.05	-34.16±0.03	-33.92±0.04	-33.79±0.04	-33.72±0.04	-33.69±0.04	-33.67±0.04
c20ng	-153.71±0.56	-151.58±0.45	-149.96±0.30	-149.10±0.21	-148.64±0.15	-148.45±0.12	-148.24±0.10
cr52	-87.57±0.64	-85.51±0.48	-83.35±0.37	-82.33±0.18	-81.93±0.13	-81.74±0.08	-81.52±0.08
cwebkb	-155.62±0.68	-153.70±0.43	-151.93±0.26	-151.00±0.17	-150.59±0.18	-150.43±0.21	-150.21±0.22
dna	-99.04±0.50	-97.80±0.17	-96.62±0.17	-96.11±0.25	-95.86±0.31	-95.77±0.33	-95.64±0.37
jester	-52.97±0.03	-52.53±0.04	-52.20±0.02	-52.03±0.01	-51.97±0.02	-51.95±0.02	-51.93±0.02
kdd	-2.20±0.03	-2.18±0.04	-2.15±0.01	-2.13±0.00	-2.13±0.00	-2.13±0.00	-2.13±0.00
kosarek	-11.11±0.04	-10.93±0.03	-10.81±0.02	-10.75±0.01	-10.73±0.01	-10.72±0.01	-10.71±0.01
msnbc	-6.39±0.04	-6.25±0.02	-6.17±0.01	-6.15±0.01	-6.15±0.01	-6.15±0.01	-6.14±0.01
msweb	-10.31±0.04	-10.03±0.04	-9.80±0.01	-9.72±0.00	-9.69±0.00	-9.69±0.00	-9.68±0.00
nltcs	-6.08±0.00	-6.03±0.00	-6.00±0.00	-6.00±0.00	-6.00±0.00	-5.99±0.00	-5.99±0.00
plants	-14.72±0.16	-13.71±0.14	-12.99±0.02	-12.65±0.03	-12.54±0.03	-12.49±0.03	-12.45±0.02
pumbs	-39.41±0.89	-33.67±0.52	-30.01±0.10	-28.50±0.04	-28.05±0.05	-27.85±0.02	-27.67±0.03
tmovie	-51.95±0.26	-50.68±0.08	-49.59±0.13	-49.12±0.08	-48.87±0.08	-48.78±0.10	-48.69±0.09
tretail	-10.96±0.04	-10.90±0.02	-10.87±0.00	-10.85±0.00	-10.85±0.00	-10.85±0.00	-10.85±0.00

Table B.2: Average $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ test log-likelihoods across 5 random seeds on 20 density estimation benchmarks. Higher is better.

Dataset	Mix-2^7	Mix-2^8	Mix-2^9	Mix-2^{10}	Mix-2^{11}	Mix-2^{12}	Mix-2^{13}
accidents	-31.00±0.15	-30.06±0.09	-29.28±0.05	-28.93±0.02	-28.79±0.01	-28.74±0.01	-28.69±0.01
ad	-16.67±0.39	-15.89±0.25	-15.27±0.08	-14.97±0.11	-14.85±0.11	-14.81±0.10	-14.76±0.10
baudio	-39.88±0.07	-39.51±0.08	-39.21±0.02	-39.08±0.02	-39.04±0.02	-39.03±0.02	-39.02±0.02
bbc	-249.01±0.94	-246.56±0.73	-244.75±0.51	-243.75±0.56	-243.27±0.54	-243.07±0.52	-242.83±0.55
bnetflix	-56.54±0.06	-56.01±0.07	-55.60±0.03	-55.42±0.02	-55.35±0.02	-55.34±0.02	-55.31±0.02
book	-34.29±0.10	-34.04±0.04	-33.90±0.03	-33.83±0.03	-33.79±0.03	-33.77±0.03	-33.75±0.03
c20ng	-151.59±0.45	-150.39±0.28	-149.28±0.17	-148.68±0.11	-148.41±0.09	-148.28±0.09	-148.17±0.09
cr52	-85.39±0.40	-83.82±0.24	-82.35±0.14	-81.70±0.09	-81.41±0.11	-81.30±0.09	-81.17±0.11
cwebkb	-150.73±0.28	-149.55±0.20	-148.59±0.21	-148.18±0.24	-147.94±0.25	-147.86±0.26	-147.77±0.26
dna	-86.79±0.25	-86.09±0.16	-85.49±0.09	-85.16±0.08	-85.01±0.07	-84.96±0.09	-84.91±0.09
jester	-52.55±0.05	-52.27±0.03	-52.06±0.02	-51.98±0.01	-51.95±0.01	-51.95±0.01	-51.94±0.01
kdd	-2.16±0.01	-2.14±0.00	-2.13±0.00	-2.12±0.00	-2.12±0.00	-2.12±0.00	-2.12±0.00
kosarek	-10.68±0.03	-10.63±0.02	-10.59±0.01	-10.57±0.01	-10.57±0.01	-10.56±0.01	-10.56±0.01
msnbc	-6.44±0.16	-6.22±0.10	-6.09±0.01	-6.07±0.00	-6.06±0.00	-6.06±0.00	-6.05±0.00
msweb	-9.89±0.05	-9.78±0.03	-9.67±0.02	-9.64±0.01	-9.63±0.01	-9.63±0.01	-9.62±0.01
nltcs	-6.05±0.02	-6.03±0.02	-6.00±0.00	-6.00±0.00	-6.00±0.00	-6.00±0.00	-5.99±0.00
plants	-13.68±0.13	-12.98±0.07	-12.53±0.02	-12.35±0.01	-12.30±0.02	-12.28±0.01	-12.26±0.01
pumbs	-26.29±0.22	-25.22±0.22	-24.26±0.07	-23.92±0.02	-23.80±0.02	-23.76±0.03	-23.71±0.03
tmovie	-51.67±0.27	-50.59±0.16	-49.90±0.12	-49.56±0.01	-49.38±0.10	-49.29±0.11	-49.23±0.10
tretail	-10.84±0.01	-10.83±0.01	-10.82±0.01	-10.82±0.01	-10.82±0.01	-10.82±0.01	-10.82±0.01

Table B.3: Latent Optimisation (LO) results for $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$. Integration points optimised for 150 epochs (with early stopping) using Adam with learning rate of 0.001.

Dataset	Mix-2^7	Mix-2^7(LO)	Mix-2^8	Mix-2^8(LO)	Mix-2^9	Mix-2^9(LO)	Mix-2^{10}	Mix-2^{10}(LO)
accidents	-31.00±0.15	-29.81±0.03	-30.06±0.09	-29.36±0.05	-29.28±0.05	-29.05±0.03	-28.93±0.02	-28.81±0.02
ad	-16.67±0.39	-15.08±0.17	-15.89±0.25	-14.73±0.13	-15.27±0.08	-14.51±0.11	-14.97±0.11	-14.42±0.09
baudio	-39.88±0.07	-39.45±0.04	-39.51±0.08	-39.25±0.03	-39.21±0.02	-39.11±0.02	-39.08±0.02	-39.04±0.02
bbc	-249.01±0.94	-245.57±0.75	-246.56±0.73	-244.03±0.74	-244.75±0.51	-243.25±0.59	-243.75±0.56	-242.79±0.58
bnetflix	-56.54±0.06	-56.02±0.03	-56.01±0.07	-55.69±0.02	-55.60±0.03	-55.48±0.02	-55.42±0.02	-55.36±0.02
book	-34.29±0.10	-33.79±0.07	-34.04±0.04	-33.69±0.08	-33.90±0.03	-33.61±0.06	-33.83±0.03	-33.55±0.02
c20ng	-151.59±0.45	-149.95±0.31	-150.39±0.28	-149.16±0.26	-149.28±0.17	-148.65±0.17	-148.68±0.11	-148.28±0.11
cr52	-85.39±0.40	-83.28±0.23	-83.82±0.24	-82.25±0.22	-82.35±0.14	-81.67±0.18	-81.70±0.09	-81.31±0.15
cwebkb	-150.73±0.28	-148.82±0.21	-149.55±0.20	-148.33±0.20	-148.59±0.21	-147.95±0.23	-148.18±0.24	-147.75±0.26
dna	-86.79±0.25	-85.64±0.24	-86.09±0.16	-85.03±0.13	-85.49±0.09	-84.78±0.12	-85.16±0.08	-84.58±0.10
jester	-52.55±0.05	-52.23±0.03	-52.27±0.03	-52.06±0.02	-52.06±0.02	-51.98±0.02	-51.98±0.01	-51.94±0.02
kdd	-2.16±0.01	-2.13±0.00	-2.14±0.00	-2.12±0.00	-2.13±0.00	-2.12±0.00	-2.12±0.00	-2.12±0.00
kosarek	-10.68±0.03	-10.60±0.01	-10.63±0.02	-10.58±0.01	-10.59±0.01	-10.56±0.01	-10.57±0.01	-10.55±0.01
msnbc	-6.44±0.16	-6.08±0.01	-6.22±0.10	-6.06±0.00	-6.09±0.01	-6.05±0.00	-6.07±0.00	-6.05±0.00
msweb	-9.89±0.05	-9.67±0.01	-9.78±0.03	-9.64±0.02	-9.67±0.02	-9.61±0.01	-9.64±0.01	-9.60±0.01
nltcs	-6.05±0.02	-6.00±0.00	-6.03±0.02	-6.00±0.00	-6.00±0.00	-5.99±0.00	-6.00±0.00	-5.99±0.00
plants	-13.68±0.13	-12.76±0.04	-12.98±0.07	-12.50±0.02	-12.53±0.02	-12.34±0.02	-12.35±0.01	-12.27±0.01
pumbs	-26.29±0.22	-24.63±0.05	-25.22±0.22	-24.17±0.07	-24.26±0.07	-23.86±0.04	-23.92±0.02	-23.70±0.02
tmovie	-51.67±0.27	-50.31±0.18	-50.59±0.16	-49.79±0.23	-49.90±0.12	-49.40±0.12	-49.56±0.01	-49.29±0.12
tretail	-10.84±0.01	-10.82±0.01	-10.83±0.01	-10.81±0.01	-10.82±0.01	-10.81±0.01	-10.82±0.01	-10.81±0.01

Table B.4: Average test-set log-likelihood on 20 density estimation benchmarks as reported on the respective papers: HCLT (Liu and Van den Broeck 2021), Strudel (Dang et al. 2020), LearnPSDD (Liang et al. 2017), Einet (Peharz et al. 2020a), LearnSPN (Gens and Domingos 2013), ID-SPN (Rooshenas and Lowd 2014), and RAT-SPN (Peharz et al. 2020b). Results for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ and $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ are computed with 2^{13} integration points. Higher is better.

Dataset	HCLT	EiNet	LearnSPN	ID-SPN	RAT-SPN	Strudel	LearnPSDD	$\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$	$\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$
accidents	**-26.78**	-35.59	-40.50	-26.98	-35.48	-29.46	-28.29	-33.27	-28.69
ad	-16.04	-26.27	-19.73	-19.00	-48.47	-16.52	-20.13	-18.72	**-14.76**
baudio	-39.77	-39.87	-40.53	-39.79	-39.95	-42.26	-41.51	**-39.02**	**-39.02**
bbc	-250.07	-248.33	-250.68	-248.93	-252.13	-258.96	-260.24	**-240.19**	-242.83
bnetflix	-56.28	-56.54	-57.32	-56.36	-56.85	-58.68	-58.53	-55.49	**-55.31**
book	-33.84	-34.73	-35.88	-34.14	-34.68	-35.77	-36.06	**-33.67**	-33.75
c20ng	-151.92	-153.93	-155.92	-151.47	-152.06	-160.77	-160.43	-148.24	**-148.17**
cr52	-84.67	-87.36	-85.06	-83.35	-87.36	-92.38	-93.30	-81.52	**-81.17**
cwebkb	-153.18	-157.28	-158.20	-151.84	-157.53	-160.50	-161.42	-150.21	**-147.77**
dna	**-79.33**	-96.08	-82.52	-81.21	-97.23	-87.10	-83.02	-95.64	-84.91
jester	-52.45	-52.56	-75.98	-52.86	-52.97	-55.30	-54.63	**-51.93**	-51.94
kdd	-2.18	-2.18	-2.18	-2.13	**-2.12**	-2.17	-2.17	-2.13	**-2.12**
kosarek	-10.66	-11.02	-10.98	-10.60	-10.88	-10.98	-10.99	-10.70	**-10.56**
msnbc	-6.05	-6.11	-6.11	-6.04	**-6.03**	-6.05	-6.04	-6.14	-6.05
msweb	-9.90	-10.02	-10.25	-9.73	-10.11	-10.19	-9.93	-9.68	**-9.62**
nltcs	-6.00	-6.01	-6.11	-6.02	-6.01	-6.06	-6.03	**-5.99**	**-5.99**
plants	-14.31	-13.67	-12.97	-12.54	-13.43	-13.72	-13.49	-12.45	**-12.26**
pumbs	-23.32	-31.95	-24.78	**-22.40**	-32.53	-25.28	-25.40	-27.67	-23.71
tmovie	-50.69	-51.70	-52.48	-51.51	-53.63	-59.47	-55.41	**-48.69**	-49.23
tretail	-10.84	-10.91	-11.04	-10.85	-10.91	-10.90	-10.92	-10.85	**-10.82**

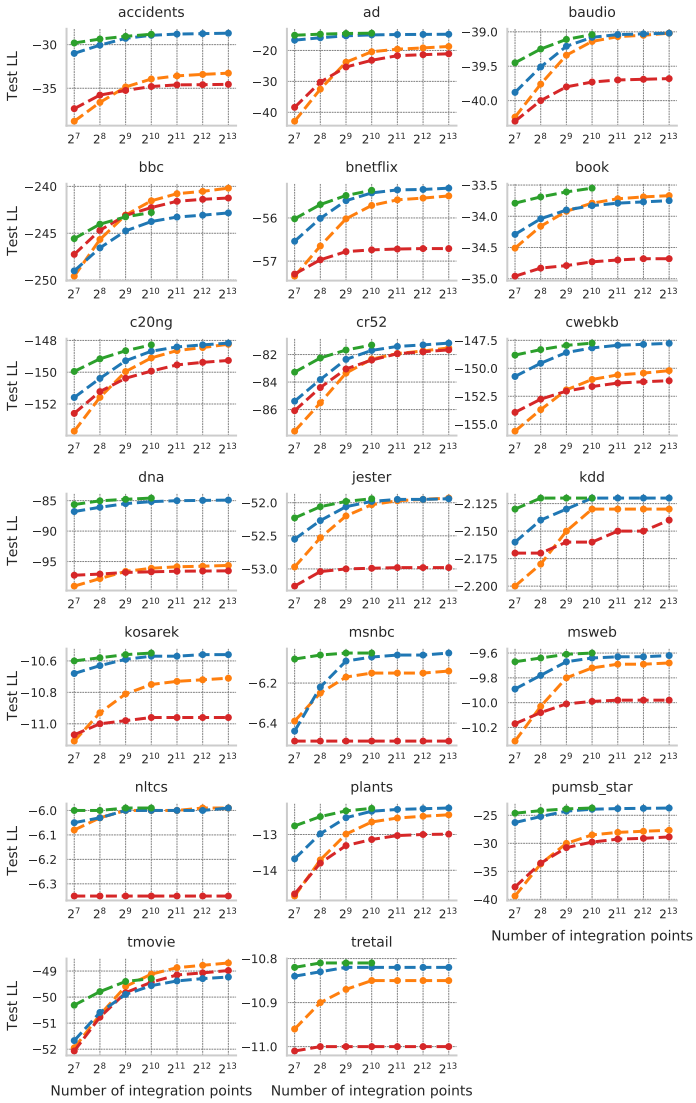

Figure B.1: Individual plots of test log-likelihood against number of integration points on 20 benchmarks for $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})$ (orange), $\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}})$ (blue), $\mathsf{LO}(\mathrm{cm}(\mathcal{G}_{\mathsf{CLT}}))$ (green) and $\mathrm{cm}(\mathcal{G}_{\mathsf{F}})_{\mathsf{VAE}}$ (red). Results are averaged over 5 random seeds.

Curriculum Vitae

Alvaro Correia was born on the 12th of November 1992 in Sao Caetano do Sul in the state of Sao Paulo, Brazil. From 2011 until 2017 he was a student of mechatronics engineering at the Universidade de Sao Paulo. During this time he also enrolled in a double degree program, studying towards a MSc in engineering at ENSTA Paris, in France, from 2014 to 2017. Before and during his PhD, he accumulated a few experiences in industry, working as an aerospace engineer at Rolls-Royce, as a machine learning researcher at Accenture and Qualcomm, and as a data scientist at Itaú Unibanco bank. In 2018, Alvaro joined Utrecht University as a PhD student under the supervision of Cassio de Campos, and in 2019 both of them moved to Eindhoven University of Technology, where Alvaro has been co-supervised by Robert Peharz. Since April 2023, Alvaro works as a machine learning researcher at Qualcomm AI Research Amsterdam.

Acknowledgements

I first begin by thanking the researchers who were instrumental in guiding me towards pursuing a PhD in the first place. Fabio Cozman, Freddy Lecue and Alexander Gepperth were present figures during my BSc and MSc degrees who taught me a lot about machine learning and research in general. Without their support and encouragement I would not have started on this path, and I am immensely grateful.

I am also extremely thankful to Cassio de Campos who took me in as his PhD student (twice) and has been a zestful advisor and friend throughout these years. I enjoyed our research discussions both on the blackboard and over a glass of beer, during which I certainly have learned plenty about machine learning, academia and research. Much if not all of the work in this thesis is fruit of Cassio's ideas and suggestions, and I am extremely thankful for his very present supervision doing my PhD.

On a similar note, I would like to thank my co-supervisor Robert Peharz. Being a leading figure in Probabilistic Circuits, he played an integral role in my PhD and, like Cassio, has been a very present and enthusiastic supervisor. I am grateful for all our interesting research discussions as well as the long hours we spent working together to meet paper deadlines.

I would also like to thank Daniel E. Worrall and Roberto Bondesan who welcomed me to their research team at Qualcomm and were great mentors and supervisors during my research internship. Our time working together was brief but extremely enriching both in terms of research ideas and professional development.

I could not go without thanking the many friends I made throughout my PhD years both in Eindhoven and Utrecht. Victor Miraldi, Vedran Kasalica, João Pizani, Lisa Bauer, Gennaro Gala, David Montalvan, Loek Tonnaer, Tim d'Hondt, Xin Du, Yulong Pei, Shiwei Liu, Laurens Stoop, Rogier Wuijts and many others

fellow researchers were an integral part of this journey. They all helped make my PhD life lighter, more productive and more fun.

A special thanks goes to my fiancée, Maria Bilkova, who has been present in my life even before the idea of doing a PhD first came to my mind and has been extremely encouraging and supportive ever since. I would not have made it without her by my side.

Finally, I would like to thank my family for their ongoing love and support despite the long distance. My parents enthusiasm about science and culture as well as their investment in my education are the main reasons I was able to find my way into research.

Alvaro Correia

SIKS Dissertations

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data

20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect

40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity

34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TUE), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems

07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection

22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges

05 Yulong Pei (TUE), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots

25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
31 Gongjin Lan (VU), Learning better – From Baby to Better
32 Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production

2021 01 Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
02 Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
07 Armel Lefebvre (UU), Research data management for open science
08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning

11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey

03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations

23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (LU), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (LU), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment

07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models