SCALABLE LEARNING APPROACHES FOR

SUM-PRODUCT-CUTSET NETWORKS

by

Tahrima Rahman

APPROVED BY SUPERVISORY COMMITTEE:

Vibhav Gogate, Chair

Latifur Khan

Vincent Ng

Nicholas Ruozzi

To my mother.

SCALABLE LEARNING APPROACHES FOR SUM-PRODUCT-CUTSET NETWORKS

by

TAHRIMA RAHMAN, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2016

ACKNOWLEDGMENTS

I would like to express my gratitude and thank a number of people in my academic, social and family life who have contributed in numerous ways towards pursuing this degree. Vibhav Gogate, my advisor, is the person I am the most indebted to for his valuable teachings, wisdom, continuous support and guidance in conducting research throughout my PhD years. I deeply appreciate his contributions towards developing a unique and collaborative research group among our fellow lab mates. His patience, friendliness and trust towards his students have made all of us in his group excel in our academic journeys and accomplishments.

I am grateful to David, Som, Li, Deepak, and Luis for being great friends. Next to my advisor's supervision, I have learned the most from them during our problem solving discussions. The team and group has been a source of inspiration and friendship. I would also like to thank the newbies in the group – Sara and Shasha – for their presence and encouraging compliments during my research presentations.

I would like to thank my committee members Dr. Latifur Khan, Dr. Vincent Ng and Dr. Nicholas Ruozzi for taking their time to evaluate my dissertation, for allowing me to audit their classes and for training excellent and talented students who have become wonderful colleagues of mine over the years.

I would like to thank my parents, my brother and his family and my husband for their encouragement and enormous support during all these years. I cannot imagine this precious journey without their help in every step.

I would like to thank NSF, DARPA, ARO and AFRL for providing all the financial support for carrying out the research.

October 2016

SCALABLE LEARNING APPROACHES FOR

SUM-PRODUCT-CUTSET NETWORKS

Publication No. _____

Tahrima Rahman, PhD
The University of Texas at Dallas, 2016

Supervising Professor: Vibhav Gogate

Probabilistic graphical models (PGMs) are widely used in practice to represent and reason about uncertainty. However, inference – process used to answer queries – in them is NP-hard in general and computationally intractable for most real-world models. To circumvent this problem, polynomial time approximate inference algorithms are routinely used in practice. Unfortunately, they are often unreliable and yield inaccurate query answers.

Therefore, recently there has been growing interest in learning tractable probabilistic models, namely models which admit polynomial time exact inference, in order to address the intractability and inaccuracy of probabilistic inference. Examples of tractable models include: thin junction trees, mixtures of trees, arithmetic circuits and sum-product networks. Although, there has been tremendous progress in algorithms that induce the structure and parameters of tractable models from data, in many real-world domains, either the accuracy of the learned models is extremely low or the computational complexity of the learning algorithms is too high, or both. Moreover, existing tractable representations use naive approaches for modeling hybrid probability distributions having both discrete and continuous variables, and as a result perform poorly in hybrid domains.

In this dissertation, we remedy the aforementioned problems by making the following contributions:

- We propose a novel tractable probabilistic model called cutset networks (CNs), and show that under certain restrictions optimal CNs can be efficiently learned from data, using only polynomial time and space. We also develop fast heuristic algorithms for learning CNs and their mixtures from data, and show that they are competitive with state-of-the-art algorithms.
- We develop novel (parallel) bagging and (sequential) boosting algorithms for learning ensembles of CNs and demonstrate via a large-scale experimental evaluation that they yield models having high accuracy (measured using the test-set log-likelihood score) in practice.
- We show that the accuracy of CNs can be further improved by combining CNs with other tractable models such as sum-product networks and arithmetic circuits yielding a new tractable model called sum-product cutset networks (SPCNs).
- We develop novel algorithms for converting tree SPCNs to more compact graph SPCNs, and show that the latter yield substantial improvements in accuracy and prediction time.
- We extend SPCNs to yield hybrid SPCNs so that they can better handle real-world domains having both discrete and continuous variables, and develop novel algorithms for learning them from data. We present results of a large empirical evaluation which show that hybrid SPCNs have substantially better predictive accuracy than existing approaches on a vast majority of real-world datasets.

TABLE OF CONTENTS

ACKNOWLEDGMENTS v
ABSTRACT vii
LIST OF FIGURES xii
LIST OF TABLES xiv
CHAPTER 1 INTRODUCTION 1
1.1 Dissertation Outline 3
CHAPTER 2 BACKGROUND 6
2.1 Notation 6
2.2 Probabilistic Graphical Models (PGMs) 6
2.2.1 Inference 8
2.3 Tractable PGMs 11
2.3.1 Tree Structured PGMs 12
2.3.2 Thin Junction Trees 12
2.3.3 Mixtures of Trees 13
2.3.4 Arithmetic Circuits 14
2.3.5 Sum-Product Networks 17
CHAPTER 3 CUTSET NETWORKS 20
3.1 Introduction 20
3.2 The Chow-Liu Algorithm for Learning Tree Distributions 21
3.3 OR Search Trees 23
3.4 Cutset Networks 24
3.5 Learning Cutset Networks 26
3.5.1 Splitting Heuristics 27
3.5.2 Termination Condition and Post-Pruning 28
3.6 Mixtures of Cutset Networks 30

3.7 Empirical Evaluation . . . 31
3.7.1 Methodology and Setup . . . 31
3.7.2 Learning Time . . . 32
3.7.3 Accuracy . . . 33
3.8 Chapter Summary . . . 35
CHAPTER 4 LEARNING ENSEMBLES OF CUTSET NETWORKS . . . 37
4.1 Introduction . . . 37
4.2 Ensembles of Cutset Networks . . . 38
4.2.1 Boosting . . . 39
4.2.2 Bagging . . . 42
4.3 Experiments . . . 44
4.3.1 Boosting Performance . . . 45
4.3.2 Bagging Performance . . . 46
4.3.3 Comparison with State-of-the-art . . . 46
4.4 Chapter Summary . . . 51
CHAPTER 5 MERGING STRATEGIES FOR SUM-PRODUCT-CUTSET NETWORKS . 54
5.1 Introduction . . . 54
5.2 Top Down Learning of SPNs . . . 56
5.3 Sum-Product-Cutset Networks . . . 60
5.4 Merging Strategies: From Trees to Graphs . . . 61
5.4.1 Our Approach . . . 62
5.4.2 Practical Merging Strategies . . . 64
5.5 Experiments . . . 66
5.5.1 Setup . . . 66
5.5.2 Algorithms Evaluated . . . 66
5.5.3 Impact of Merging on Test Set Log-Likelihood . . . 68
5.5.4 Comparison with State-Of-The-Art . . . 68
5.6 Chapter Summary . . . 71
CHAPTER 6 LEARNING HYBRID SUM-PRODUCT-CUTSET NETWORKS . . . 73
6.1 Introduction . . . 73

6.2 Background . . . 74
6.3 Representation . . . 76
6.3.1 Hybrid Cutset Networks . . . 76
6.3.2 Hybrid Sum-Product Cutset Networks (HSPCNs) . . . 78
6.4 Learning HSPCNs . . . 78
6.4.1 Learning Tree Structured CLGs . . . 79
6.4.2 Decomposition . . . 84
6.4.3 Splitting . . . 84
6.5 Experiments . . . 85
6.5.1 Density Estimation . . . 86
6.5.2 Classification . . . 87
6.6 Chapter Summary . . . 88
CHAPTER 7 CONCLUSION . . . 89
7.1 Contributions . . . 89
7.1.1 Proposed Tractable PGMs . . . 89
7.1.2 Proposed Learning Algorithms . . . 90
7.2 Future Work . . . 91
7.2.1 Discriminative Learning of CNs . . . 91
7.2.2 Dynamic Discritization of Continuous Variables in Hybrid Models . . . 92
7.2.3 Learning More Expressive Base Models . . . 93
7.2.4 Relational Merging . . . 93
REFERENCES . . . 95
VITA

LIST OF FIGURES

2.1 (a) A Bayesian network and (b) a Markov network over four variables $\boldsymbol{X} = \{A, B, C, D\}$. The domain of each variable is $\{0, 1\}$. Each variable in the Bayesian network has an associated CPT. The Markov network has two potentials, one defined over the scope $\{A, B, C\}$ and the second defined over the scope $\{A, B, D\}$. An example potential $\phi(A, B, C)$ is shown. 8

2.2 A Markov network over variables $\{A, B, C, D, E\}$ (left) and its junction tree (right), $J_T = (\boldsymbol{C}_T, \boldsymbol{S}_T)$. The width of the junction tree is $3 - 1 = 2$. The reader can verify that the treewidth of the model is 2. 11

2.3 A mixture of trees model with three tree Bayesian networks T_1, T_2 and T_3 weighted by ω_1, ω_2 and ω_3 respectively over the set of variables $\{A, B, C, D\}$. Figure (a) is a mixture with identically structured (shared structure) trees and Figure (b) is a mixture with different structured trees. 14

2.4 (a)Bayesian network over binary variables X and Y. (b) An arithmetic circuit representing the joint distribution over variables X and Y in (a) and the network polynomial function f. Each λ is an indicator variable taking values from $\{0, 1\}$ and each real value is a parameter in the distribution.. 15

2.5 Bottom-up computation performed by the AC in Figure 2.4 for evidence X=0. Blue colored indicators have values set to 1 while red colored indicators have values set to 0. 16

2.6 An SPN over binary discrete variables $\boldsymbol{X} = \{A, B, C\}$. Each variable in $\boldsymbol{X}$ is defined over the domain $\{0, 1\}$. A $+$ denotes a sum node while a $\times$ denotes a product node. The leaves are the Bernoulli random variables with their parameters. 17

3.1 (a) A probability distribution, (b) An OR tree representing the distribution given in (a). The left branch from a node represents conditioning by 0, whereas the right branch represents conditioning by 1. 23

3.2 The OR tree in Figure 3.1 computing the probability (0.336) of the MAP tuple $\langle b = 0, c = 1\rangle$ given evidence $a = 1$. 24

3.3 A cutset network over binary random variables $\boldsymbol{X} = \{a, b, c, d, e, f\}$. Left branches in the OR tree represent conditioning by 0 while right branches represent conditioning by 1. 25

5.1 An SPCN rooted by a latent-sum node (denoted by a $+$) over discrete variables $\boldsymbol{X} = \{A, B, C, D, E\}$. All observed and latent variables are over binary domains and left edges represent conditioning by value 0 or false and right edges represent conditioning by value 1 or true. A and B are the only observed cutset variables. Product nodes (denoted by $\times$) decompose the model differently and each leaf node (denoted by a rectangle) is a cutset network (CN) over the remaining observed variables (appears in the subscript as a comma separated list). 61

5.2 Three example SPNs over variables $\{V_1, V_2, V_3, V_4\}$. We are assuming that all variables are binary and take values from the domain $\{0, 1\}$. Leaf nodes express univariate distributions. For example, the node $V_2 : 0.6$ expresses the probability distribution $P(V_2 = 1) = 0.6$. Sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. (a) Tree SPN (SPN which is a rooted directed acyclic tree) that decomposes according to a tree Markov network $V_4 - V_3 - V_1 - V_2$. (b) Graph SPN that is equivalent to the tree SPN given in (a) obtained by merging identical sub-trees. (c) Graph SPCN over latent and observed sum nodes. 62

5.3 Figure demonstrating how to simplify and thus reduce the size of the SPCN after merging. As before, sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. $S_{1,2}$ is an SPCN obtained by merging SPCNs S_1 and S_2. (a): shows how the SPCN can be reduced when the two child nodes of an observed sum node are merged. The node $V_i : 0.3$ represents a univariate probability distribution over V_i with $P(V_i = 1) = 0.3$. (b): shows how the SPCN can be reduced when the two child nodes of a latent sum node are merged. 65

6.1 A tree CLG over discrete variable X_1 ranging over domain $\{0, 1\}$ and continuous variables Y_1 and Y_2. 76

6.2 (a)Hybrid TAN and (b)Hybrid BMN and (c) Hybrid CN over discrete variables $\{X_1, X_2, Class\}$ and continuous variables $\{Y_1, Y_2\}$ 77

6.3 An HSPCN rooted by a latent-sum node over discrete variables $\{X_1, X_2, X_3\}$ and continuous variables $\{Y_1, Y_2, Y_3, Y_4, Y_5\}$. X_1 is the only observed sum node. Each product node decomposes the model differently and each leaf is a tree CLG over remaining observed variables. 78

LIST OF TABLES

3.1 Runtime Comparison (in seconds). Time-limit for each algorithm: 48 hours. †indicates that the algorithm did not terminate in 48 hours. 33

3.2 Average test set log-likelihood. †indicates that the results of this algorithm are not available. 34

3.3 Head-to-head comparison of the number of wins (in terms of average test set log-likelihood score) achieved by one algorithm (row) over another (column), for all pairs of the 6 best performing algorithms used in our experimental study. 35

4.1 Test set log-likelihood scores of boosting algorithms. Winning scores are bolded and underlines highlight GBDE over BDE. FD:= Fixed Depth and VD:=Variable Depth. . . 47

4.2 Learning time comparison of boosting methods. FD:= Fixed Depth and VD:=Variable Depth. 48

4.3 Average test set log-likelihood scores and learning time (in seconds) of bagged ensembles of cutset networks (CNs). FD:= Fixed Depth CNs (without randomized variable selection), FD_R:= Fixed Depth CNs with randomized variable selection, VD:= Variable Depth CNs (without randomized variable selection) and VD_R:= Variable Depth CNs with randomized variable selection. Bold values indicate the best values achieved for the dataset. 49

4.4 Average test set log-likelihood scores and learning time (in seconds) of bagged ensembles of small mixtures of cutset networks (MCNs). FD:= Fixed Depth MCNs (without randomized variable selection), FD_R:= Fixed Depth MCNs with randomized variable selection, VD:= Variable Depth MCNs (without randomized variable selection) and VD_R:= Variable Depth MCNs with randomized variable selection. Bold values indicate the best values achieved for the dataset. 50

4.5 Test set log-likelihood comparison. (Ties are not bolded). 52

4.6 Runtime comparison (in seconds). †indicates that the algorithm did not terminate in 48 hrs. 53

5.1 Examples of SPN structure learning approaches in the literature that follow the prescription given in Algorithm 4. Base case is the stopping criteria for the recursive algorithm. 59

5.2 Table showing the impact of merging on the average test-set log likelihood, time complexity and prediction time of L-SPCNs and O-SPCNs (all values rounded to two decimal places). We use the following notation: (1) T-LL: Average test-set log likelihood for the tree SPCNs; (2) G-LL: average test-set log likelihood for the graph SPCNs obtained from the tree SPCNs by merging similar sub-SPCNs; (3) |T|: number of parameters in the tree SPCN; (4) |G|: number of parameters in the graph SPCN; (5) CR:=Compression Ratio = $\frac{|T|}{|G|}$; (6) T-Time: Tree SPCN learning time in seconds and (7) G-Time: Time in seconds required by the merging algorithm (thus the total learning time for graph SPCN is T-time+G-time seconds). In each row, **bold** values indicate the best score for each of the two SPCN categories: L-SPCN and O-SPCN. . 69

5.3 Average test set log-likelihood comparison with state-of-the-art tractable model learners. **Bold** values indicate the winning score for the corresponding dataset. T-LL: Bagged LL of tree SPCNs and G-LL: Bagged LL of graph SPCNs. Column "Best-LL to date" gives the best log-likelihood score to date for each dataset obtained using the following competing approaches: ID-SPN (Rooshenas and Lowd, 2014), ACMN (Lowd and Rooshenas, 2013), MCN (Rahman et al., 2014), SPN-SVD (Adel et al., 2015), and ECN (Rahman and Gogate, 2016a). 71

6.1 Structural comparison of various hybrid models. HNB:=Hybrid Naive Bayes, HBMN:= Hybrid Bayesian Multinet, HCN:= Hybrid Cutset Network, HSPN:= Hybrid Sum-Product Network, and HSPCN:= Hybrid Sum-Product-Cutset Network. 79

6.2 Comparison of average test set log-likelihood scores. Columns $|Train|$, $|Valid|$ and $|Test|$ provides the number of samples for training, validation and test set respectively. Columns labeled D and C indicate the number of discrete and continuous valued variables present in the domain respectively. Bold values indicate the highest average score achieved by a model. 87

6.3 Comparison of classification accuracy evaluated on the test set. Bold values indicate the best accuracy achieved by the model on the particular instance. 88

CHAPTER 1

INTRODUCTION

Solving important tasks in real-world application domains such as computer vision and natural language processing requires easy access to tools that can efficiently represent and reason about uncertainty. For example, in topic modeling, in order to associate topics with a document, we need to represent and reason about uncertainty in how various words in the document, either individually or as a collection, influence the possible topics. Similarly, in order to automatically label images (e.g., outdoor, indoor, etc.), we have to model the uncertainty in how various objects or pixels in the image affect its possible labels.

Probability theory is a popular approach for representing and reasoning about uncertainty in a mathematically precise manner. The main assumption is that the world consists of random variables and a probability distribution over them represents complete knowledge about the world. Thus, in order to find the topics associated with a document, an approach based on probability theory would represent words and topics as random variables, and then define a probability distribution that models uncertainty in how words and topics affect other words and topics. Unfortunately, the three main tasks in probabilistic modeling: (1) representing – storing the probability distribution on a computer using only polynomial space, (2) inferring – answering queries defined over the distribution, and (3) learning – inducing a probability distribution from data, are all computationally hard.

Probabilistic graphical models (PGMs) such as Bayesian and Markov networks exploit, and in many cases make conditional independence assumptions in order to yield a compact, polynomial space representation of a joint probability distribution defined over a large number of random variables. In PGMs, conditional independence relations (assumptions) are compactly represented using a directed or an undirected graph; vertices in the graph represent random variables while edges

represent direct dependencies between corresponding variables (thus lack of edges denote conditional independence). However, although PGMs require only polynomial space, inference over them can be quite challenging. In particular, marginal inference, the task of computing the probability of a random variable given observations, is #P-hard, while maximum-a-posteriori (MAP) inference, the task of finding an assignment of values to all variables where the maximum probability, is NP-hard. As a result, approximate inference methods such as loopy belief propagation and Gibbs sampling are widely used in practice. Unfortunately, they often yield highly inaccurate and high variance estimates, leading to poor predictive performance.

One approach to tackle the inaccuracy and unreliability of approximate inference is to use so-called tractable models such as thin junction trees (Bach and Jordan, 2001), tractable arithmetic circuits (ACs) (Darwiche, 2003), probabilistic sentential decision diagrams (Kisa et al., 2014), AND/OR decision diagrams (Dechter and Mateescu, 2007; Mateescu et al., 2008), sum-product networks (SPNs) (Poon and Domingos, 2011) and mixtures of trees (Meila and Jordan, 2000). Exact inference in these models can be performed in time that scales polynomially (often linearly) with the size of the model and therefore the complexity and accuracy of inference is no longer an issue. Interestingly, experimental results in numerous recent studies have shown that the performance of approaches that learn tractable models from data is similar or better than approaches that learn Bayesian and Markov networks from data. These results suggest that *controlling exact inference complexity* is the key to superior end-to-end performance.

Despite these promising results, a key bottleneck remains. Barring a few exceptions, algorithms that learn the structure and parameters of tractable models from data are computationally expensive, requiring several hours for even moderately sized problems. For instance, approaches presented in (Gens and Domingos, 2013; Lowd and Rooshenas, 2013; Rooshenas and Lowd, 2014) need more than "10 hours" of CPU time for datasets having 200 variables and $10,000$ examples. There are several reasons for this, with the main reason being the high computational complexity of *conditional independence tests*. For example, the LearnSPN algorithm of Gens and Domingos (Gens and Domingos, 2013) and the ID-SPN algorithm of Rooshenas and Lowd (Rooshenas

and Lowd, 2014) for learning tractable sum-product networks, spend a substantial amount of their execution time on partitioning the given set of variables into conditionally independent components. Other algorithms with strong theoretical guarantees, such as learning efficient Markov networks (Gogate et al., 2010), and learning thin junction trees (Bach and Jordan, 2001; Chechetka and Guestrin, 2008; Narasimhan and Bilmes, 2004) also suffer from the same problem.

In this dissertation, we propose a new tractable PGM, called *cutset networks (CNs)*, which are rooted OR search trees with tree Bayesian or Markov networks at the leaves. Cutset networks derive their name from Pearl's cutset conditioning method (Pearl, 1988). The key idea in cutset conditioning is to condition on a subset of variables in the graphical model, called the *cutset*, such that the remaining network has a tree structure. Since, exact probabilistic inference can be performed in time that is linear in the size of the tree (using Belief propagation (Pearl, 1988) for instance), the complexity of cutset conditioning is exponential in the cardinality (size) of the cutset. If the cutset is bounded, then cutset conditioning is tractable. However, unlike classic cutset conditioning, cutset networks can take advantage of determinism (Chavira and Darwiche, 2008; Gogate and Domingos, 2010b) and context-specific independence (Boutilier et al., 1996) by allowing different variables to be conditioned on at the same level in the OR search tree. As a result, these models can be very diverse and can yield a compact representation even if the size of the cutset is arbitrarily large.

1.1 Dissertation Outline

The rest of the dissertation is organized as follows. We begin by conducting literature review on several existing tractable PGMs in Chapter 2. Then in subsequent chapters we address each of the following issues/questions related to learning CNs through contributions made to date:

- **Why Cutset Networks and How to learn them from data?**

  The key advantage of cutset networks is that they admit efficient, polynomial time learning

algorithms. This is because only the leaf nodes, which represent tree distributions, take advantage of conditional independence, while the OR search tree does not. As a result, to learn cutset networks from data, we do not have to run expensive conditional independence tests at any internal node in the OR tree. Moreover, the leaf distributions can be learned in polynomial time, using the classic Chow-Liu algorithm (Chow and Liu, 1968). As a result, if we assume that the size of the cutset (or the height of the OR tree) is bounded by k, and given that the time complexity of the Chow-Liu algorithm is $O(n^2|\boldsymbol{\mathcal{D}}|)$, where n is the number of variables and $|\boldsymbol{\mathcal{D}}|$ is the number of training examples, the optimal cutset network with a static ordering of the cutset variables can be learned in $O(n^{(k+2)}|\boldsymbol{\mathcal{D}}|)$ time.

In Chapter 3, we formally describe the semantics of CNs and propose our first algorithm to learn these models from data. Experimental results on several real-world datasets show that CNs can achieve state-of-the-art accuracy (measured by test data log-likelihood) orders of magnitude faster than other algorithms.

- **Learning ensembles of CNs to improve the accuracy**

  In Chapter 4, we consider generalized additive mixtures of cutset networks and develop sequential boosting-based and parallel bagging-based approaches for learning them from data leveraging vast amount of previous work on boosting and bagging algorithms (cf. (Zhou, 2012)) as well as their generalizations for density estimation (Rosset and Segal, 2002; Ridgeway, 2002; Welling et al., 2002). We perform and report on a comprehensive empirical evaluation, comparing our new algorithms with several state-of-the-art systems. Our empirical evaluation clearly demonstrates the power of our new approaches.

- **Learning graph structured sum-product-cutset networks**

  Sum-product networks (SPNs) are tractable PGMs with a rooted directed acyclic graph of alternating levels of latent variable mixtures (latent sum nodes) and decompositions (product nodes) with univariate distributions attached to the leaves. In Chapter 5, we propose

sum-product-cutset networks (SPCNs) which combine SPNs and CNs by introducing cutset variables (observed sum nodes) in the latent sum-product space of SPNs and tree Bayesian or Markov networks at the leaves. SPCNs have more representational power than CNs since mixture models can better express complex distributions. We can learn SPCNs by combining the steps of CN and SPN learning algorithms. The structure learning algorithm for CNs, SPNs or SPCNs induce tree structured models – a single conditioning path from the root node to each leaf node. When learned from small number of training examples, trees suffer from high variance and easily overfit. We propose to develop post-learning approaches that induce graph structured models by merging similar sub structures. The key benefits of graph representations over tree representations include (1) smaller computational complexity which facilitates faster online inference, and (2) better generalization accuracy because of reduced variance, at the cost of slight increase in the learning time.

- **How can we extend these models to continuous domains?**

  In Chapter 6, we extend SPCNs to model joint probability distributions over both discrete and continuous variables. Hybrid SPCNs have a sum-product space over a combination of a subset of discrete observed and latent variables and tree structured conditional linear Gaussian Bayesian networks (Murphy, 1998) at the leaves. We propose a generative algorithm for learning the structure and parameters of these hybrid networks given data over a mixed domain. The proposed algorithm is general enough to learn a variety of models including hybrid tree-augmented naive Bayes, hybrid Bayesian multinets, hybrid cutset networks and hybrid sum-product networks. Our experimental results show that hybrid SPCNs are significantly better in modeling the underlying probability distribution as well as the decision boundaries than other popular density estimators and classifiers.

In the final chapter, we summarize our research contributions mentioned in this dissertation and propose several potential future work in the field of learning tractable sum-product-cutset networks.

CHAPTER 2

BACKGROUND

In this chapter, we present background on probabilistic graphical models as well as their tractable counterparts. We begin by formalizing the common notation used throughout this dissertation.

2.1 Notation

We use the following convention for denoting sets of variables and assignments of values to them. Let $\boldsymbol{X}$ denote a set of n discrete random variables where each variable $v \in \boldsymbol{X}$ takes a value from a finite discrete domain Δ_v. Let $\boldsymbol{\Delta}$ denote the set of all domains and $\boldsymbol{\Delta_X}$ denote the Cartesian product of the domains of all variables in $\boldsymbol{X}$, namely $\boldsymbol{\Delta_X} = \prod_{v \in \boldsymbol{X}} \Delta_v$. We will denote values in Δ_v by x_v, namely $x_v \in \Delta_v$, and an assignment of values (full or complete assignment) to all n variables in $\boldsymbol{X}$ by $\boldsymbol{x}$, namely $\boldsymbol{x} \in \boldsymbol{\Delta_X}$. Let $\boldsymbol{A} \subseteq \boldsymbol{X}$, then $\boldsymbol{x_A}$ denotes the projection of complete assignment $\boldsymbol{x}$ onto $\boldsymbol{A}$ and $\boldsymbol{\Delta_A}$ denotes the Cartesian product of the domains of variables in $\boldsymbol{A}$.

2.2 Probabilistic Graphical Models (PGMs)

Probabilistic graphical models (PGMs) (Koller and Friedman, 2009; Pearl, 1988; Darwiche, 2009) combine graphs and probability theory to compactly encode a joint probability distribution over a large number of random variables. A PGM $\mathcal{G}$ can be described by the tuple $\langle \boldsymbol{X}, \boldsymbol{\Delta}, G, \boldsymbol{\mathcal{F}} \rangle$ where G=($\boldsymbol{X}$, $\boldsymbol{E}$) is a graph and $\boldsymbol{\mathcal{F}}$ is a set of real-valued functions defined over subsets of variables in $\boldsymbol{X}$. Each node in G represents a variable in $\boldsymbol{X}$ and each edge represents a direct probabilistic interaction between two variables, the parameters of which are quantitatively described by a function in $\boldsymbol{\mathcal{F}}$ that the two variables appear in. Each function $f \in \boldsymbol{\mathcal{F}}$ is defined over a subset of variables called the $scope$ of f and maps each possible configuration of those variables to a non-negative

value i.e. $f : \boldsymbol{x}_{scope(f)} \rightarrow \mathbb{R}^+$. A PGM $\mathcal{G}$ factorizes the joint probability distribution over $\boldsymbol{X}$ as a product of the functions in $\boldsymbol{\mathcal{F}}$:

$$P(\boldsymbol{x}) = \frac{1}{Z} \prod_{f \in \boldsymbol{\mathcal{F}}} f(\boldsymbol{x}_{scope(f)})$$

where $Z = \sum_{\boldsymbol{x} \in \boldsymbol{\Delta}_{\boldsymbol{X}}} \prod_{f \in \boldsymbol{\mathcal{F}}} f(\boldsymbol{x}_{scope(f)})$ is the normalizing constant. Edges in G can be directed as in Bayesian networks or undirected as in Markov networks.

Bayesian Networks

A Bayesian network (BN) is a PGM: $\mathcal{G}_{\mathcal{B}} = \langle \boldsymbol{X}, \boldsymbol{\Delta}, G_{\mathcal{B}}, \boldsymbol{P} \rangle$ with the following two properties: (1) $G_{\mathcal{B}} = (\boldsymbol{X}, \boldsymbol{E}_{\mathcal{B}})$ is a directed acyclic graph (DAG) with nodes labeled by variables in $\boldsymbol{X}$; and (2) each variable $v \in \boldsymbol{X}$ is associated with a function in $\boldsymbol{P}$ that defines the conditional probability $\theta_{x_v|\boldsymbol{x}_{\pi(v)}}$ for each possible value x_v of v given each possible assignment $\boldsymbol{x}_{\pi(v)}$ to its parents $\pi(v) \subseteq \boldsymbol{X}$ and hence called the *conditional probability table* or CPT of v. Thus, the set $\boldsymbol{P}$ contains n CPTs that describe the parameters of the Bayesian network. A Bayesian network factorizes the probability distribution over $\boldsymbol{X}$ as:

$$P(\boldsymbol{x}) = \prod_{v \in \boldsymbol{X}} \theta_{\boldsymbol{x}_v|\boldsymbol{x}_{\pi(v)}}$$

Figure 2.1(a) shows an example Bayesian network.

Markov Networks

A probabilistic graphical model $\mathcal{G}_{\mathcal{M}} = \langle \boldsymbol{X}, \boldsymbol{\Delta}, G_{\mathcal{M}}, \boldsymbol{F} \rangle$ is called a Markov network (MN) when $G_{\mathcal{M}} = (\boldsymbol{X}, \boldsymbol{E}_{\mathcal{M}})$ is an undirected graph with the set of nodes $\boldsymbol{X}$ and the set of undirected edges $\boldsymbol{E}_{\mathcal{M}}$. $\boldsymbol{F} = \{\phi_1, ..., \phi_m\}$ is the set of $potential$ functions where each function ϕ_i maps each instantiation of the variables in its scope to a non-negative real number. Potential functions represent the compatibility of the variables in the model. There exists an undirected edge in $\boldsymbol{E}_{\mathcal{M}}$ between two variables u and v if they appear together in the scope of a function $\phi_i \in \boldsymbol{F}$. The probability

distribution represented by a Markov network is computed as:

$$P(\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{m} \phi_i(\boldsymbol{x}_{scope(\phi_i)})$$

Figure 2.1(b) shows an example Markov network.

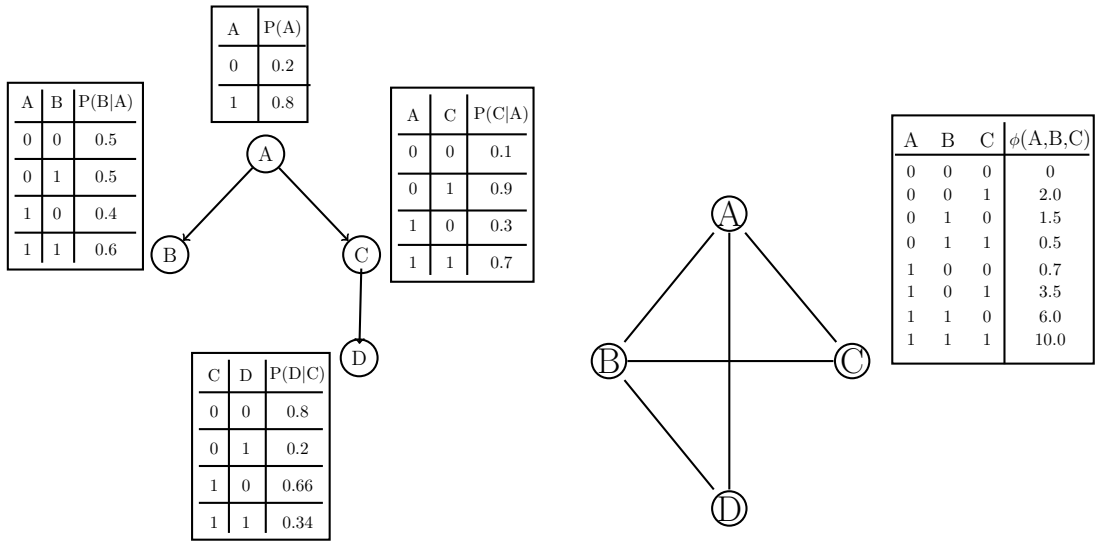


Figure 2.1. (a) A Bayesian network and (b) a Markov network over four variables $\boldsymbol{X} = \{A, B, C, D\}$. The domain of each variable is $\{0, 1\}$. Each variable in the Bayesian network has an associated CPT. The Markov network has two potentials, one defined over the scope $\{A, B, C\}$ and the second defined over the scope $\{A, B, D\}$. An example potential $\phi(A, B, C)$ is shown.

2.2.1 Inference

Since Bayesian and Markov networks represent joint distributions over random variables, one can perform various inference tasks over them which involve computing answers to probabilistic queries. Let $\mathbf{E} \subseteq \mathbf{X}$ be the set of *observed* or *evidence* variables and $\mathbf{Q} = \mathbf{X} \setminus \mathbf{E}$ be the set of *unobserved* or *non-evidence* variables. Let $\boldsymbol{e}$ be a complete assignment to the evidence variables and $\boldsymbol{q} \in \boldsymbol{\Delta_Q}$ be a complete assignment to the non-evidence variables. We call $\boldsymbol{e}$ the *evidence*. The

composition of two or more partial assignments is performed by an operator $\kappa()$ which basically concatenates the assignments into a complete assignment. The various inference tasks that can be performed using a Bayesian or Markov network are:

- Probability of Evidence (**PE**)

  The PE task is to compute the probability of evidence, which is defined as:

  $$P(\boldsymbol{e}) = \frac{1}{Z} \sum_{\boldsymbol{q} \in \boldsymbol{\Delta_Q}} P(\kappa(\boldsymbol{q}, \boldsymbol{e}))$$

  Here $Z = 1$ for Bayesian networks. This task basically sums over all possible assignments to the non-evidence variables $\boldsymbol{Q}$ from the joint probability distribution.

- Posterior Marginal Probability (**MAR**)

  MAR is the task of finding the marginal probability distribution of a variable $v \in \boldsymbol{Q}$ given $\boldsymbol{e}$. It is defined as:

  $$P(x_v|\boldsymbol{e}) = \frac{P(\kappa(x_v, \boldsymbol{e}))}{P(\boldsymbol{e})} \; \forall x_v \in \Delta_v \tag{2.1}$$

  Both the numerator and the denominator of equation 2.1 is a **PE** task. Therefore any algorithm that can perform the **PE** task can also be used to perform the **MAR** task.

- Maximum-a-Posteriori (**MAP**)

  The goal of **MAP** inference is to find an assignment to all the non-evidence variables that has the highest probability given evidence $\boldsymbol{e}$. Mathematically,

  $$\underset{\boldsymbol{q} \in \boldsymbol{\Delta_Q}}{\operatorname{argmax}} P(\boldsymbol{q}|\boldsymbol{e}) = \underset{\boldsymbol{q} \in \boldsymbol{\Delta_Q}}{\operatorname{argmax}} \frac{P(\kappa(\boldsymbol{q}, \boldsymbol{e}))}{P(\boldsymbol{e})} \tag{2.2}$$

  The denominator in 2.2 can be disregarded as a normalizing constant and hence the maximization problem becomes equivalent to $\underset{\boldsymbol{q} \in \boldsymbol{\Delta_Q}}{\operatorname{argmax}} P(\kappa(\boldsymbol{q}, \boldsymbol{e}))$.

- Marginal Maximum-a-Posteriori (**MMAP**)

  The goal of **MMAP** inference is to find an assignment to a subset $\boldsymbol{A} \subset \boldsymbol{Q}$ of non-evidence variables given evidence $\boldsymbol{e}$. It is defined as:

$$\underset{\boldsymbol{a} \in \boldsymbol{\Delta_A}}{\operatorname{argmax}} P(\boldsymbol{a}|\boldsymbol{e}) = \underset{\boldsymbol{a} \in \boldsymbol{\Delta_A}}{\operatorname{argmax}} P(\kappa(\boldsymbol{a}, \boldsymbol{e})) = \underset{\boldsymbol{a} \in \boldsymbol{\Delta_A}}{\operatorname{argmax}} \sum_{\boldsymbol{b} \in \boldsymbol{\Delta_B}} P(\kappa(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{e}))$$

  where $\boldsymbol{B} = \boldsymbol{Q} \setminus \boldsymbol{A}$.

All the above inference tasks involve a series of summations, multiplications and/or maximization over an exponential number of states of some variables, and hence naively enumerating over those states to compute an answer is intractable (Roth, 1996). In particular, **PE** and **MAR** tasks are $\#P$-hard in the worst case (Cooper, 1990), **MAP** is NP-complete, and **MMAP** is NP^{PP}-complete (Park, 2002).

Using certain compiled representations of the PGM, inference can be carried out in a more efficient manner. These representations exploit structural properties of the PGM to reduce the exponential time and space overhead of inference. *Junction trees* (Lauritzen and Spiegelhalter, 1988), also known as join-trees or cluster-trees are one such representation and are constructed from a given ordering of the variables. Nodes in a junction tree are called *clusters* and edges are called *separators*. Both the clusters and separators are labeled with variables; the labels are such that they satisfy the following *running intersection property*: for each variable v, the set of clusters and separators that mention v form a connected sub-tree of the junction tree. Any arbitrary distribution P($\boldsymbol{X}$) can be represented using a junction tree $J_T = (\boldsymbol{C_T}, \boldsymbol{S_T})$, where $\boldsymbol{C}_T$ is the set of clusters and $\boldsymbol{S}_T$ is the set of separators (see Figure 2.2). Let $V(c)$ where $c \in \boldsymbol{C}_T$ be the label of cluster c (namely $V(c) \subseteq \boldsymbol{X}$) and $V(s)$ where $s \in \boldsymbol{S}_T$ be the label of separator s. Then, the junction tree represents the following probability distribution:

$$P(\boldsymbol{x}) = \frac{\prod_{c \in \boldsymbol{C}_T} P(\boldsymbol{x}_{V(c)})}{\prod_{s \in \boldsymbol{S}_T} P(\boldsymbol{x}_{V(s)})}$$

Once a junction tree is constructed, either from a PGM or learned directly from data, **PE**, **MAR**, and **MAP** inference tasks can be solved in time that is bounded exponentially by the maximum cluster size, defined as $\max_{c \in \boldsymbol{C}_T} V(c)$. Thus, the largest cluster in the junction tree dominates the complexity of inference. This gives rise to an important measure of the complexity of inference, called the *treewidth*, defined as follows. Given a PGM $\mathcal{G}$ and an ordering of its variables o, the size of the largest cluster minus one of $\mathcal{G}$'s junction tree along o is called the $width$ of o and treewidth is the smallest width over all such orderings. The complexity of inference is exponential in the treewidth plus one of the model.

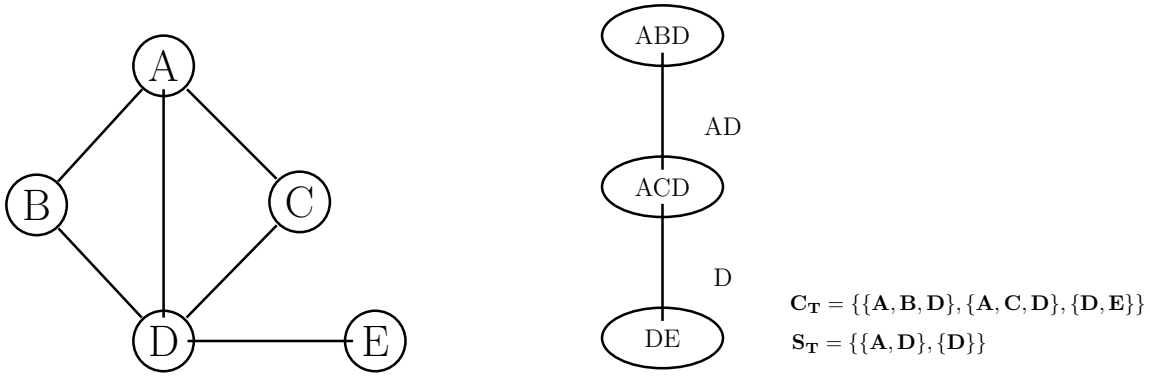


Figure 2.2. A Markov network over variables $\{A, B, C, D, E\}$ (left) and its junction tree (right), $J_T = (\boldsymbol{C}_T, \boldsymbol{S}_T)$. The width of the junction tree is $3 - 1 = 2$. The reader can verify that the treewidth of the model is 2.

2.3 Tractable PGMs

Tractable probabilistic graphical models comprise a restricted subclass of probabilistic models which admit polynomial time exact inference. From an inference point of view, tractable models are preferred because they obviate the need for approximate inference methods, which are often inaccurate and unreliable. From a learning point of view, tractable models are preferred because they are less prone to overfitting. In the next five subsections, we briefly review several popular tractable probabilistic models relevant to our research.

2.3.1 Tree Structured PGMs

A tree Bayesian network is a Bayesian network in which each variable has at most one parent while a tree Markov network is a Markov network whose primal (or interaction) graph is a tree. The primal graph of graphical model is an undirected graph which has a vertex for each variable in the model and an edge between any two vertices that appear together in the scope of a function. It is known that both tree Bayesian and Markov networks have the same representation power and therefore can be used interchangeably. The treewidth of a tree Bayesian or Markov network is 1 or equivalently the size of the largest cluster of its junction tree is 2 leading to exact inference complexity to be $\mathcal{O}(nd^2)$ where d is the maximum domain size. Tree distributions can be learned efficiently using the Chow-Liu algorithm (Chow and Liu, 1968). The algorithm will be discussed in detail in Section 3.2.

2.3.2 Thin Junction Trees

As mentioned earlier, junction trees are representations for efficiently carrying out inference tasks. Unfortunately, learning optimal junction trees, even under the restriction that their width is bounded by two is an NP-hard task (Srebro, 2003). Therefore, we have to settle for approximate methods.

(Bach and Jordan, 2001) propose an approximate algorithm that is generalization of the Chow-Liu algorithm for learning junction trees that have treewidth larger than one but still small enough to be tractable for exact inference (such low treewidth models ensure that inference will be computationally feasible in practice). The learned models are called *thin junction trees*. (Chechetka and Guestrin, 2008; Narasimhan and Bilmes, 2004) proposed algorithms having probably approximately correct (PAC) guarantees for learning thin junction trees. However, these algorithms have high polynomial complexity and are practical only when the treewidth is bounded by 4. Because of this, they often yield highly biased models having low accuracy and are seldom used in practical applications as a result.

2.3.3 Mixtures of Trees

(Meila and Jordan, 2000) proposed mixtures of trees models which represent the joint probability distributions over a set of discrete variables by weighted collections of tree structured PGMs e.g. tree Bayesian or Markov networks. A mixture of trees $\mathcal{M}$ is represented by a tuple $\langle \mathbf{T}, \boldsymbol{\omega} \rangle$, where $\mathbf{T} = \{\mathrm{T}_1, ..., \mathrm{T}_K\}$ is a collection of K tree Bayesian (Markov) networks and $\boldsymbol{\omega} = \{\omega_1, ..., \omega_K\}$ is a set of K real valued normalized weights i.e. $\sum_k \omega_k = 1$. Each tree $\mathrm{T}_i \in \mathbf{T}$ is called a *mixture component* and is associated with weight $\omega_i \in \boldsymbol{\omega}$ called the *mixture coefficient*.

Given a mixtures of trees $\mathcal{M} = \langle \mathbf{T}, \boldsymbol{\omega} \rangle$, the probability of a full assignment $\boldsymbol{x}$ is given by:

$$P(\boldsymbol{x}; \mathcal{M}) = \sum_{i=1}^{K} \omega_k \mathrm{T}_k(\boldsymbol{x})$$

where $\mathrm{T}_k(\boldsymbol{x})$ is the probability of $\boldsymbol{x}$ computed by the k^{th} tree. As shown in Figure 2.3, a mixture of trees can be understood as a model having a single discrete latent (unobserved) variable h with domain values $\{1, ..., K\}$ such that the probability of h taking on a value k is ω_k. Conditioned on h, the resulting model is a tree Bayesian (Markov) network over the observed variables. Hence, the *posterior marginal* probability of $h = k$ given evidence $\boldsymbol{x}$ is:

$$P(h = k|\boldsymbol{x}) = \frac{\omega_k \mathrm{T}_k(\boldsymbol{x})}{\sum_{i=1}^{K} \omega_i \mathrm{T}_i(\boldsymbol{x})}$$

Mixture components can share the same structure (Figure 2.3 (a)) or have different structures (Figure 2.3 (b)).

PE, **MAR** and **MAP** inference tasks can be solved in polynomial time on mixtures of trees. However, since the latent variable h is typically uninterpretable, a more relevant task is a type of **MMAP** inference, in which we seek the most likely assignment to the interpretable variables $\boldsymbol{Q}$ given evidence on $\boldsymbol{E}$ where $\boldsymbol{X} = \boldsymbol{Q} \cup \boldsymbol{E}$. Unfortunately, this MMAP inference query is at least NP-hard in mixtures of trees and we have to resort to approximate inference algorithms such as variational approximations (Liu and Ihler, 2013) for answering it.

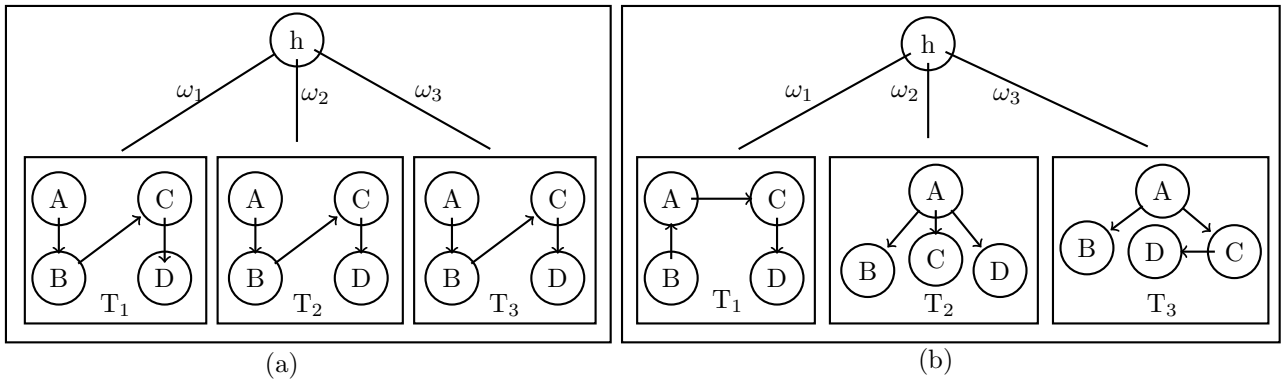


Figure 2.3. A mixture of trees model with three tree Bayesian networks T_1, T_2 and T_3 weighted by ω_1, ω_2 and ω_3 respectively over the set of variables $\{A, B, C, D\}$. Figure (a) is a mixture with identically structured (shared structure) trees and Figure (b) is a mixture with different structured trees.

2.3.4 Arithmetic Circuits

Arithmetic circuits (ACs) (Darwiche, 2001, 2003) are compiled representations of PGMs. These circuits are rooted directed acyclic graphs with internal nodes representing arithmetic operations like sums and products and leaf nodes representing states of variables ($indicators$) and parameters in the model. ACs can be viewed as machines for computing the *network polynomial* – function that represents the probability distribution induced by the PGM (Figure 2.4). The network polynomial has a term for each possible configuration of the variables and hence is exponential in size. But ACs can take advantage of symmetries like context-specific independence and determinism to compactly represent the network polynomial. The advantage of compiling a Bayesian or a Markov network into an arithmetic circuit is that once it is compiled it can be used repeatedly for different inference tasks.

Inference in ACs

During probabilistic inference using an AC, indicators at the leaves are set according to the evidence and computation is performed in a bottom-up fashion from these inputs to the output at the

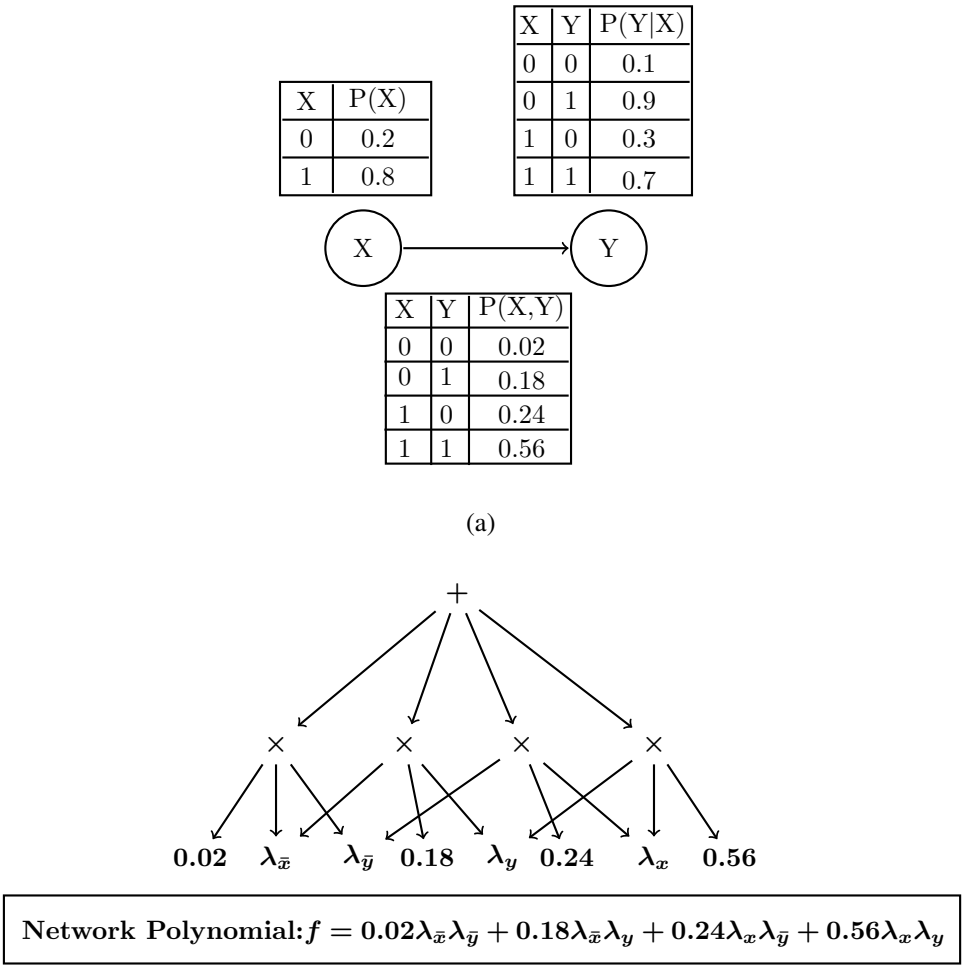


Figure 2.4. (a)Bayesian network over binary variables X and Y. (b) An arithmetic circuit representing the joint distribution over variables X and Y in (a) and the network polynomial function f. Each λ is an indicator variable taking values from $\{0, 1\}$ and each real value is a parameter in the distribution..

root as shown in Figure 2.5. Each sum node computes the summation of the values from its children while each product node computes the product of the values of its children. The presence of multiple paths from the root to a node allows an AC to take advantage of caching by computing the sub-circuits only once and then using its results many times. ACs can compute MAP values and MAP assignments by converting each internal sum node into a max node. The *size* of an AC is the number of edges it contains and this defines a measure of its complexity and also the complexity

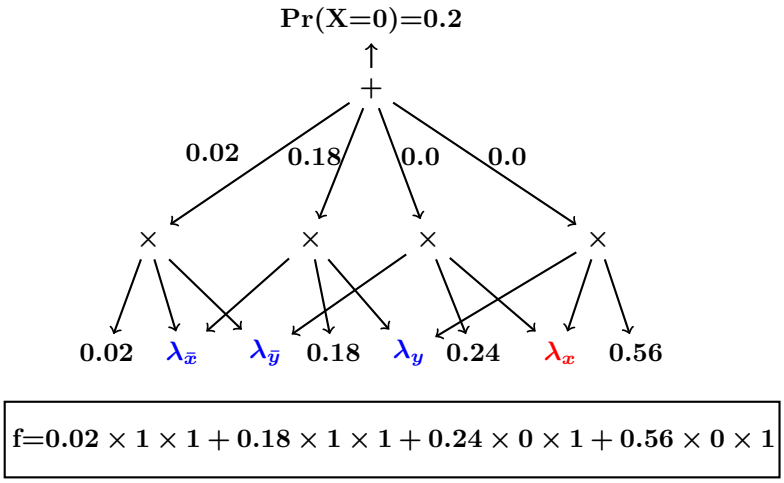


Figure 2.5. Bottom-up computation performed by the AC in Figure 2.4 for evidence X=0. Blue colored indicators have values set to 1 while red colored indicators have values set to 0.

of performing inference using it – the time and space complexity of performing inference is thus linear in the size of the AC.

Learning Tractable Models Using ACs

Lowd and Domingos (Lowd and Domingos, 2008) proposed to learn ACs over observed variables by using the AC size as a learning (inductive) bias within a Bayesian network structure learning algorithm, and then compiling the induced Bayesian network to an AC. Lowd and Rooshenas (Rooshenas and Lowd, 2013) extended this algorithm to learn Markov networks having small AC size. The latter performs much better in terms of test set log likelihood score than the former because of the increased flexibility afforded by the undirected Markov network structure. (Rooshenas and Lowd, 2013) improved upon their work by learning mixtures of small ACs. Recently Rooshenas and Lowd (Rooshenas and Lowd, 2016) proposed a discriminative structure learning algorithm for ACs which generates more compact structures than generatively learned ACs.

2.3.5 Sum-Product Networks

Sum-product networks (SPNs) (Poon and Domingos, 2011) are recently proposed deep probabilistic architectures. Similar to ACs, SPNs are rooted directed acyclic graphs with alternating levels of sum and product nodes and univariate distributions at the leaves (see Figure 2.6 for example). Sum nodes are equivalent to latent variables representing mixtures of two or more sub-SPNs while product nodes represent decomposition of the model. The set of variables appearing in an SPN defines its *scope*. Each child of a sum node is defined over the same scope while children of a product node have disjoint scopes. Formally an SPN can be recursively defined as follows (Gens and Domingos, 2012).

- A tractable univariate distribution is an SPN.
- A product of SPNs with disjoint scopes is an SPN.
- A weighted sum of SPNs with the same scope is an SPN.
- Nothing else is an SPN.

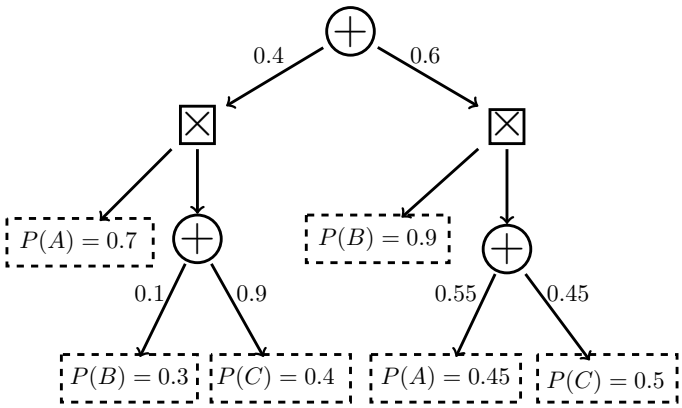


Figure 2.6. An SPN over binary discrete variables $\boldsymbol{X} = \{A, B, C\}$. Each variable in $\boldsymbol{X}$ is defined over the domain $\{0, 1\}$. A $+$ denotes a sum node while a $\times$ denotes a product node. The leaves are the Bernoulli random variables with their parameters.

An SPN represents a (normalized) probability distribution when the weights attached to each sum node sum to one. Any unnormalized SPN can be normalized in linear time.

Inference in SPNs

Similar to ACs, variables are instantiated according to the evidence $\boldsymbol{e}$ in an SPN during probabilistic inference. Each sum node $\mathcal{S}$ computes the probability of evidence as the weighted summation of the probabilities computed by its children (product nodes) as follows.

$$\mathcal{S}(\boldsymbol{e}) = \sum_{\mathcal{P}_i \in Ch(\mathcal{S})} \omega(\mathcal{S}, \mathcal{P}_i)\mathcal{P}_i(\boldsymbol{e})$$

where $Ch(\mathcal{S})$ is the children of $\mathcal{S}$, each of which is a product node $\mathcal{P}_i$ and $\omega(\mathcal{S}, \mathcal{P}_i)$ is the weight attached to the i^{th} child of $\mathcal{S}$. Each product node $\mathcal{P}$ computes the probability of evidence as the unweighted product of the probabilities of its children (sum nodes).

$$\mathcal{P}(\boldsymbol{e}) = \prod_{\mathcal{S}_i \in Ch(\mathcal{P})} \mathcal{S}_i(\boldsymbol{e})$$

The complexity of computing probability of evidence using an SPN is linear in the size (number of edges) of the SPN.

Learning SPNs

There has been intensive research in learning the structure and parameters of SPNs from data. (Poon and Domingos, 2011) proposed a generative parameter learning algorithm for SPNs under the assumption that the structure of the SPN, namely a deep architecture having alternating levels of latent sum and product nodes, is given. (Gens and Domingos, 2012) extended this approach to yield a discriminative parameter learning algorithm.

Recently, (Gens and Domingos, 2013) proposed a top-down recursive algorithm that alternates between partitioning the set of training instances and the set of variables to learn both the structure and parameters of SPNs. (Peharz et al., 2013) proposed to learn SPNs by clustering variables in a greedy bottom-up manner. (Rooshenas and Lowd, 2014) proposed to learn SPNs with ACs representing tractable markov networks at the leaves while (Vergari et al., 2015) proposed to learn SPNs with tree structured PGMs at the leaves. (Adel et al., 2015) proposed a faster learning algorithm

for SPNs by extracting rank-1 sub-matrices. Their proposed method works for both generative and discriminative settings. (Nath and Domingos, 2015) generalized SPN to the relational framework and proposed the first algorithm for learning tractable statistical relational models.

The key advantage of SPNs and other equivalent representations such as ACs over thin-junction trees is that they can be much compact and never larger than the latter. This is because they take advantage of various fine-grained structural properties such as determinism, context-specific independence, dynamic variable orderings and caching (cf. (Darwiche, 2003; Chavira and Darwiche, 2007; Dechter and Mateescu, 2007; Gogate and Domingos, 2010b)). For instance, in some cases, they can represent high-treewidth junction trees using only a handful of sum and product nodes (Poon and Domingos, 2011).

CHAPTER 3

CUTSET NETWORKS[1]

3.1 Introduction

In this chapter, we introduce cutset networks (CNs) – a new tractable probabilistic model for representing a joint probability distribution, defined over a large set of *discrete* random variables. We describe the semantics of CNs and propose an efficient polynomial time algorithm for learning their structure and parameters from data. We also present a method for learning mixtures of CNs, which significantly improves the accuracy of CNs on datasets having a large number of variables (high-dimensional data) but relatively few examples.

The rest of the chapter is organized as follows. In Section 3.2, we review the classic Chow-Liu algorithm (Chow and Liu, 1968) for inducing optimal tree structured PGMs from data. Section 3.3 discusses OR search trees, which graphically describe the search space explored during *probabilistic inference by conditioning*. Both tree PGMs and OR search trees are the basic building blocks of CNs. We formally introduce CNs in Section 3.4 and then describe a simple and efficient recursive algorithm for learning their structure and parameters from data in Section 3.5. Section 3.6 presents a method for learning mixtures of CNs. In Section 3.7, we present experimental results on learning CNs from several real-world high-dimensional datasets and compare their performance to other state-of-the-art tractable model learners. We conclude in Section 3.8.

The research in this chapter is based on (Rahman et al., 2014).

[1]We acknowledge the contribution of Prasanna P. Kothalkar in running, collecting and formatting the experimental results presented in this chapter.

3.2 The Chow-Liu Algorithm for Learning Tree Distributions

The Chow-Liu algorithm (Chow and Liu, 1968) is a classic algorithm for learning optimal tree structured discrete probabilistic graphical models from data. Specifically, if $P(\boldsymbol{X})$ is an arbitrary joint probability distribution over a set of variables $\boldsymbol{X}$, then the Chow-Liu algorithm approximates $P(\boldsymbol{X})$ by a tree distribution $T(\boldsymbol{X})$ such that T is *optimal*, namely there does not exist a tree distribution T' that has smaller approximation error (measured using the KL-divergence) than T.

A tree distribution can be specified using an undirected tree $G_T = (\boldsymbol{X}, \boldsymbol{E}_T)$, namely a Markov network, with the following parameters: (1) each edge $(u, v) \in \boldsymbol{E}_T$ is associated with the joint probability distribution over $\{u, v\}$, namely $T(\boldsymbol{x}_u, \boldsymbol{x}_v)$; and (2) each vertex $v \in \boldsymbol{X}$ is associated with the marginal probability distribution over the corresponding variable v, namely $T(\boldsymbol{x}_v)$. The (tree) Markov network represents the following probability distribution:

$$T(\boldsymbol{x}) \;=\; \frac{\prod\limits_{(u,v)\in \boldsymbol{E}_T} T(\boldsymbol{x}_u, \boldsymbol{x}_v)}{\prod\limits_{v\in \boldsymbol{X}} T(\boldsymbol{x}_v)^{deg(v)-1}}$$

where $deg(v)$ is the degree of vertex v or the number of incident edges to v. If G_T is a directed model such as a Bayesian network, then

$$T(\boldsymbol{x}) \;=\; \prod_{v\in \boldsymbol{X}} T(\boldsymbol{x}_v|\boldsymbol{x}_{\pi(v)})$$

where $T(\boldsymbol{x}_v|\boldsymbol{x}_{\pi(v)})$ is the conditional probability of v given its parents $\pi(v)$ such that $|\pi(v)| \leq 1$. The Kullback-Leibler divergence (Kullback and Leibler, 1951) $KL(P, T)$ between $P(\boldsymbol{X})$ and $T(\boldsymbol{X})$ is defined as:

$$KL(P, T) \;=\; \sum_{\boldsymbol{x}} P(\boldsymbol{x}) \log \left(\frac{P(\boldsymbol{x})}{T(\boldsymbol{x})} \right)$$

In order to minimize $KL(P, T)$, Chow and Liu proved that each selected edge $(u, v) \in \boldsymbol{E}_T$ has to maximize the total mutual information, $\sum_{(u,v)\in \boldsymbol{E}_T} I(u, v)$. Mutual information, denoted by $I(u, v)$, is a measure of mutual dependence between two random variables u and v and is given by:

$$I(u, v) \;=\; \sum_{x_u\in\Delta_u} \sum_{x_v\in\Delta_v} P(x_u, x_v) \log \left(\frac{P(x_u, x_v)}{P(x_u)P(x_v)} \right) \tag{3.1}$$

To maximize $\sum_{(u,v)\in \boldsymbol{E}_T} I(u,v)$, the Chow-Liu procedure computes the mutual information $I(u,v)$ for all possible pairs of variables in $\boldsymbol{X}$ and then finds the *maximum weighted spanning tree* (Cormen, 2009) $G_T = (\boldsymbol{X}, \boldsymbol{E}_T)$ such that each edge $(u,v) \in \boldsymbol{E}_T$ is weighted by $I(u,v)$. The marginal distribution $T(u,v)$ of a pair of variables (u,v) connected by an edge is the same as $P(u,v)$. Let δ_{max} be the largest domain size. Then the time complexity of computing the $\frac{n(n-1)}{2}$ pairwise mutual information values is $O(n^2\delta_{max}^2)$ and the complexity of finding the maximum weighted spanning tree (by Kruskal's algorithm for example) is $O(n^2 \log n)$. Therefore, the time complexity of approximating a tree distribution $T(\boldsymbol{X})$ from $P(\boldsymbol{X})$ is $O(n^2\delta_{max}^2 + n^2 \log n)$.

The task of learning the maximum-likelihood tree distribution from data $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^N\}$ can be defined as follows:

$$T^* = \underset{T}{\operatorname{argmax}} \sum_{\boldsymbol{x}^i \in \boldsymbol{\mathcal{D}}} \log T(\boldsymbol{x}^i)$$

which is maximized when $KL(\hat{P}||T)$ is minimized where $\hat{P}$ is the empirical distribution. As before, the Chow-Liu procedure weighs edges between each pair of variables (u,v) by the empirical mutual information $\hat{I}(u,v)$ computed as:

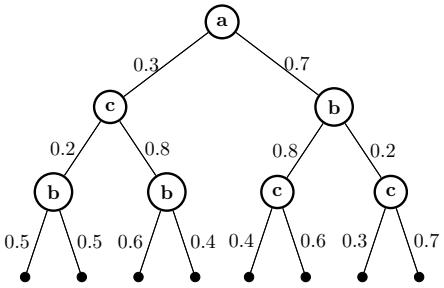
$$\hat{I}(u,v) = \sum_{x_u \in \Delta_u} \sum_{x_v \in \Delta_v} \hat{P}(x_u, x_v) \log \frac{\hat{P}(x_u, x_v)}{\hat{P}(x_u)\hat{P}(x_v)}$$

which is maximized when the marginals $\hat{P}(x_u, x_v)$, $\hat{P}(x_u)$ and $\hat{P}(x_v)$ are estimated by the standard maximum-likelihood technique that requires a single pass over the N samples. The complexity of estimating the empirical mutual information values between all pairs of variables is $O(n^2N)$. Therefore, the time complexity of estimating the structure and parameters of a tree distribution from data using the Chow-Liu algorithm is $O(n^2\delta_{max}^2 + n^2 \log(n) + n^2N)$. In practice, $N > \log(n)$ and $N > \delta_{max}^2$, and therefore the practical time complexity of the Chow-Liu algorithm is $O(n^2N)$.

Tree distributions are attractive because: (1) learning both the structure and parameters of the distribution are tractable; (2) several probabilistic inference tasks can be solved in linear time ; and (3) they have intuitive interpretations.

a	b	c	P(a,b,c)
0	0	0	0.030
0	0	1	0.144
0	1	0	0.030
0	1	1	0.096
1	0	0	0.224
1	0	1	0.336
1	1	0	0.042
1	1	1	0.098

(a)



(b)

Figure 3.1. (a) A probability distribution, (b) An OR tree representing the distribution given in (a). The left branch from a node represents conditioning by 0, whereas the right branch represents conditioning by 1.

3.3 OR Search Trees

OR trees are rooted trees which are used to represent the search space explored during probabilistic inference by conditioning (Pearl, 1988; Dechter and Mateescu, 2007). Each node in an OR tree is labeled by a variable v in the model. Each edge emanating from a node represents the conditioning of the variable v at that node by a value $x_v \in \Delta_v$ and is labeled by the marginal probability of the variable-value assignment given the path from the root to the node. For simplicity, we will focus on binary valued variables. For binary variables, assume that left edges represent the assignment of variable v to 0 and right edges represent $v = 1$. A similar representation can be used for multi-valued variables.

Any distribution can be represented using an OR Tree. In the worst-case, the tree will require $O(2^{n+1})$ parameters to specify the distribution. Figure 3.1 shows a probability distribution and a possible OR tree.

The distribution represented by an OR tree $\mathcal{O}$ is given by:

$$P(\boldsymbol{x}) = \prod_{(u,v) \in path_{\mathcal{O}}(\boldsymbol{x})} \omega(u, v) \tag{3.2}$$

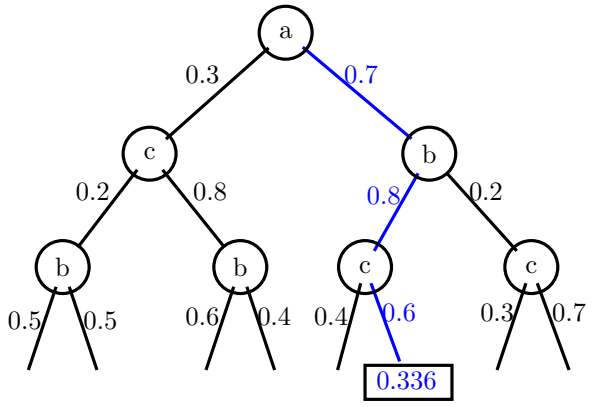


Figure 3.2. The OR tree in Figure 3.1 computing the probability (0.336) of the MAP tuple $\langle b = 0, c = 1 \rangle$ given evidence $a = 1$.

where $path_{\mathcal{O}}(\boldsymbol{x})$ is the path from the root to a unique leaf node corresponding to the assignment $\boldsymbol{x}$ and $\omega(u, v)$ is the probability value attached to the edge between the OR nodes u and v. The *size* of an OR tree is measured by the number of its edges. During probabilistic inference, the tree is traversed in a depth-first manner starting from the root and visiting each edge in the tree consistent with the evidence. Nodes labeled by non-evidence variables compute the weighted summation of the values returned by its children which is equivalent to summing out that variable. Similarly, the MAP probability can be computed when each OR node functions as *max* node instead of a sum node as shown in Figure 3.2. Samples from an OR tree can be generated by sampling edges (variable-value pairs) based on their probabilities starting from the root. Therefore, the complexity of inference and sampling is linear in the size of the tree.

3.4 Cutset Networks

Cutset Networks (CNs) are a hybrid of rooted OR trees and tree distributions, with an OR tree at the top and a tree distribution attached to each leaf node of the OR tree. Formally, a cutset network is a pair $\mathcal{C} = (\mathcal{O}, \boldsymbol{T})$ where $\mathcal{O}$ is a rooted OR tree and $\boldsymbol{T} = \{T_1, \ldots, T_{\mathcal{L}}\}$ is a collection of tree

distributions. The distribution represented by a cutset network is given by:

$$P(\boldsymbol{x}) = \left(\prod_{(u,v) \in path_{\mathcal{O}}(x)} \omega(u,v) \right) \left(T_{\ell(\boldsymbol{x})}(\boldsymbol{x}_{Scope(T_{\ell(\boldsymbol{x})})}) \right) \tag{3.3}$$

where $path_{\mathcal{O}}(x)$ is the path from the root to the unique leaf node $\ell(\boldsymbol{x})$ corresponding to the assignment $\boldsymbol{x}$, $\omega(u,v)$ is the probability value attached to the edge between the OR nodes u and v and $T_{\ell(\boldsymbol{x})}$ is the tree distribution associated with $\ell(\boldsymbol{x})$ and $Scope(T_{\ell(\boldsymbol{x})})$ is the set of variables over which $T_{\ell(\boldsymbol{x})}$ is defined. Figure 3.3 shows an example cutset network.

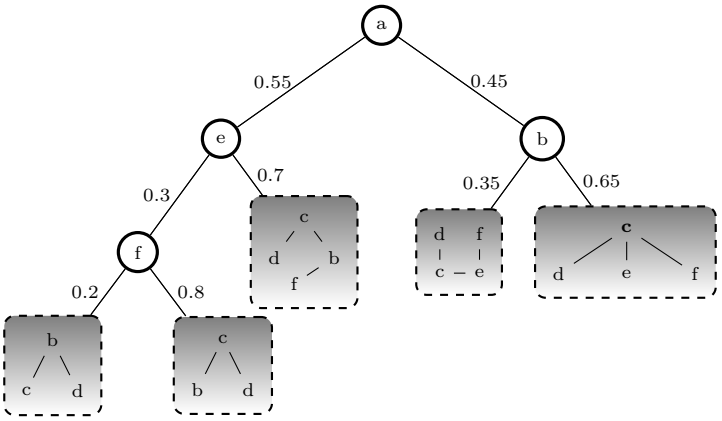


Figure 3.3. A cutset network over binary random variables $\boldsymbol{X} = \{a, b, c, d, e, f\}$. Left branches in the OR tree represent conditioning by 0 while right branches represent conditioning by 1.

CNs can represent a variety of models, for example, a CN of depth zero is a tree distribution and a CN of depth one is a Bayesian multinet (Geiger and Heckerman, 1996b).Note that unlike classic cutset conditioning, cutset networks can take advantage of determinism (Chavira and Darwiche, 2008) and context-specific independence (Boutilier et al., 1996) by branching on different variables at the same level (or depth) (Gogate and Domingos, 2010b; Gogate et al., 2010). As a result, they can yield a compact representation, even if the size of the cutset[2] is arbitrarily large. For example, consider the cutset network given in Figure 3.3. The left most leaf node represents a tree Bayesian network over $\boldsymbol{X} \setminus \{a, e, f\}$ while the right most leaf node represents a tree Bayesian network over

[2]Given a graph $G = (V, E)$, $C \subseteq V$ is a cutset of G if the subgraph over $V \setminus C$ is a tree.

$\boldsymbol{X} \setminus \{a, b\}$. Technically, the size of the cutset can be as large as the union of the variables mentioned at various levels in the OR tree. Thus, for the cutset network given in Figure 3.3, the size of the cutset can be as large as $\{a, b, e, f\}$.

3.5 Learning Cutset Networks

If we bound the number of nodes in the OR tree of the cutset network by k, then the optimal cutset network can be learned in $O(n^{k+2}k\delta_{max}N)$ time where n is the number of variables, δ_{max} is the maximum domain size and N is the number of examples. To see this, notice that there are at most n choices for each node and thus we will have to evaluate $O(n^k)$ OR trees; each OR tree will have at most $O(k\delta_{max})$ leaves and the Chow-Liu algorithm at each leaf node requires $O(n^2N)$ time. Although, the algorithm, just described is tractable, it has high polynomial complexity and is infeasible for any reasonable k (e.g., 10-100) that we would like to use in practice.

Therefore, to make our algorithm practical, we use splitting heuristics and pruning techniques developed over the last few decades for inducing decision trees from data (Quinlan, 1986; Mitchell, 1997). The splitting heuristics help us quickly learn a reasonably good cutset network, without any backtracking, while the pruning techniques such as pre-pruning and reduced-error (post) pruning help us avoid overfitting. To improve the accuracy further, we also consider mixtures of cutset networks, which generalize mixtures of Chow-Liu trees (Meila and Jordan, 2000) and develop an expectation-maximization algorithm for learning them from data.

Simply put, given training data $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ defined over a set $\boldsymbol{X}$ of variables, we can use the following recursive or divide-and-conquer approach to learn a cutset network from $\boldsymbol{\mathcal{D}}$ (see Algorithm 1). Select a variable using the given *splitting heuristic*, place it at the root and make one branch for each of its possible values. Repeat the approach at each branch, using only those instances that reach the branch. If at any time, some pre-defined *termination condition* is satisfied, run the Chow-Liu algorithm on the remaining data and variables. It is easy to see that the optimal probability value, assuming that we are using the maximum likelihood estimation

Algorithm 1: LearnCN ($\boldsymbol{\mathcal{D}}$, $\boldsymbol{X}$)

Input: Training dataset $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^N\}$, Variables $\boldsymbol{X}$.
Output: A cutset network $\mathcal{C}$

if Termination *condition* is satisfied **then**
  return ChowLiuTree($\boldsymbol{\mathcal{D}}$, $\boldsymbol{X}$)
end if
Heuristically select a variable $v \in \boldsymbol{X}$ for splitting
Create a new node O_v labeled by v.
/*
Each node has a left child $O_v.left$, a right child $O_v.right$, a left probability $O_v.lp$ and a right probability $O_v.rp$.
*/
Let $\boldsymbol{\mathcal{D}}_{v=0} = \{\boldsymbol{x}^i \in \boldsymbol{\mathcal{D}} | \boldsymbol{x}^i_v = 0\}$
Let $\boldsymbol{\mathcal{D}}_{v=1} = \{\boldsymbol{x}^i \in \boldsymbol{\mathcal{D}} | \boldsymbol{x}^i_v = 1\}$
$O_v.lp \leftarrow \frac{|\boldsymbol{\mathcal{D}}_{v=0}|}{|\boldsymbol{\mathcal{D}}|}$
$O_v.rp \leftarrow \frac{|\boldsymbol{\mathcal{D}}_{v=1}|}{|\boldsymbol{\mathcal{D}}|}$
$O_v.left \leftarrow$ LearnCN ($\boldsymbol{\mathcal{D}}_{v=0}, \boldsymbol{X} \setminus v$)
$O_v.right \leftarrow$ LearnCN ($\boldsymbol{\mathcal{D}}_{v=1}, \boldsymbol{X} \setminus v$)
return O_v

principle, attached to each branch in the OR tree is the fraction of the instances at the parent that actually reach the branch.

The two main choices in the above algorithm are which variable to split on and the termination condition. We discuss each of them in turn, next, followed by the algorithm to learn mixtures of cutset networks from data.

3.5.1 Splitting Heuristics

Intuitively, we should split on a variable that reduces the expected entropy (or the information content) of the two partitions of the data created by the split. The hope is that when the expected entropy is small, we will be able to represent it faithfully using a simple distribution such as a tree Bayesian network. Unfortunately, unlike traditional classification problems, in which we are interested in (the entropy of) a specific class variable, estimating the joint entropy of the data when

the class variable is not known is a challenging task (we just don't have enough data to reliably measure the joint entropy). Therefore, we propose to approximate the joint entropy by the average entropy over individual variables. Formally, for our purpose, the entropy of data $\boldsymbol{\mathcal{D}}$ defined over a set $\boldsymbol{X}$ of variables is given by:

$$\widehat{H}(\boldsymbol{\mathcal{D}}) = \frac{1}{|\boldsymbol{X}|} \sum_{v \in \boldsymbol{X}} H_{\boldsymbol{\mathcal{D}}}(v) \tag{3.4}$$

where $H_{\boldsymbol{\mathcal{D}}}(v)$ is the entropy of variable v relative to $\boldsymbol{\mathcal{D}}$. It is given by:

$$H_{\boldsymbol{\mathcal{D}}}(v) = -\sum_{x_v \in \Delta_v} P(x_v) log(P(x_v))$$

Given a closed-form expression for the entropy of the data, we can calculate the information gain or the expected reduction in the entropy after conditioning on a variable v using the following expression:

$$Gain_{\mathcal{D}}(v) = \widehat{H}(\boldsymbol{\mathcal{D}}) - \sum_{x_v \in \Delta_v} \frac{|\boldsymbol{\mathcal{D}}_{x_v}|}{|\boldsymbol{\mathcal{D}}|} \widehat{H}(\boldsymbol{\mathcal{D}}_{x_v})$$

where $\boldsymbol{\mathcal{D}}_{x_v} = \{\boldsymbol{x}^i \in \boldsymbol{\mathcal{D}} | \boldsymbol{x}^i_v = x_v\}$.

From the discussion above, the splitting heuristic is obvious: select a variable that has the highest information gain.

3.5.2 Termination Condition and Post-Pruning

A simple termination condition that we can enforce is stopping when the number of examples at a node falls below a fixed threshold. Alternatively, we can also declare a node as a leaf node if the entropy falls below a threshold. Unfortunately, both of these criteria are highly dependent on the threshold used. A large threshold will yield shallow OR trees that are likely to underfit the data (high bias) while a small threshold will yield deep trees that are likely to overfit the data (high variance). To combat this, inspired by the decision tree literature (Quinlan, 1986, 1993), we propose to use reduced error pruning. In reduced error pruning, we grow the tree fully and post-prune in a bottom-up fashion. (Alternatively, we can also prune in a top-down fashion).

The benefits of pruning over using a fixed threshold are that it avoids the horizon effect (the thresholding method suffers from lack of sufficient look ahead). Pruning comes at a greater computational expense than threshold based stopped splitting and therefore for problems with large training sets, the expense can be prohibitive. For small problems, though, these computational costs are low and pruning should be preferred over stopped splitting. Moreover, pruning is an anytime method and as a result we can stop it at any time.

Formally, our proposed reduced error pruning for cutset networks operates as follows. We divide the data into two sets: training data and validation data. Then, we build a full OR tree over the training data, declaring a node as a leaf node using a weak termination condition (e.g., the number of examples at a node is less than or equal to 5). Then, we recursively visit the tree in a bottom up fashion, and replace a node and the sub-tree below it by a leaf node (namely, a tree distribution) if it increases the log-likelihood of the validation set.

We summarize the time and space complexity of learning (using Algorithm 1) and inference in cutset networks in the following theorem.

Theorem 1. *The time complexity of learning cutset networks is $O(n^2Ns)$ where s is the number of nodes in the cutset network, N is the number of training samples and n is the number of variables. The space complexity of the algorithm is $O(ns)$, which also bounds the space required by the cutset networks. The time complexity of performing marginal and maximum-a-posteriori inference in a cutset network is $O(ns)$.*

Proof. The time complexity of computing the gain at each internal OR node is $O(n^2N)$. Similarly, the time complexity of running the Chow-Liu algorithm at each leaf node is $O(n^2N)$. Since there are s nodes in the cutset network, the overall time complexity is $O(n^2Ns)$. The space required to store an OR node is $O(1)$ while the space required to store a tree Bayesian network is $O(n)$. Thus, the overall space complexity is $O(max(n, 1)s) = O(ns)$. The time complexity of performing inference at each leaf Chow-Liu node is $O(n)$ while inference at each internal OR node can be done in constant time. Since the tree has s nodes, the overall inference complexity is $O(ns)$. □

3.6 Mixtures of Cutset Networks

Similar to mixtures of trees (Meila and Jordan, 2000), we define mixtures of cutset networks (MCNs) as distributions of the form:

$$P(\boldsymbol{x}) = \sum_{i=1}^{k} \lambda_i \mathcal{C}_i(\boldsymbol{x}) \tag{3.5}$$

with $\lambda_i \geq 0$ for $i = 1, \ldots, k$, and $\sum_{i=1}^{k} \lambda_i = 1$. Each mixture component $\mathcal{C}_i(\boldsymbol{x})$ is a cutset network and λ_i is its mixture co-efficient. At a high level, one can think of the mixture as containing a latent variable z which takes a value $i \in \{1, \ldots, k\}$ with probability λ_i.

Next, we present a version of the expectaton-maximization algorithm (EM) (Dempster et al., 1977) for learning mixtures of cutset networks from data. The EM algorithm operates as follows. We begin with random parameters. At each iteration t, in the expectation-step (E-step) of the algorithm, we find the probability of completing each training example, using the current model. Namely, for each training example $\boldsymbol{x}^j$ and each component i, we compute

$$P^t(z = i|\boldsymbol{x}^j) = \frac{\lambda_i^t \mathcal{C}_i^t(\boldsymbol{x}^j)}{\sum_{r=1}^{k} \lambda_r^t \mathcal{C}_r^t(\boldsymbol{x}^j)}$$

Then, in the maximization-step (M-step), we learn each mixture component i, using a weighted training set in which each example $\boldsymbol{x}^j$ has weight $P^t(z = i|\boldsymbol{x}^j)$. This yields a new mixture component $\mathcal{C}_i^{t+1}$. In the M-step, we also update the mixture co-efficients using the following expression:

$$\lambda_i^{t+1} = \frac{\sum_{j=1}^{N} P^t(z = i|\boldsymbol{x}^j)}{N}$$

We can run EM until it converges or until a pre-defined bound on the number of iterations is exceeded. The quality of the local maxima reached by EM is highly dependent on the initialization used and therefore in practice, we typically run EM using several different initializations and choose parameter settings having the highest log-likelihood score. Notice that by varying the number of mixture components, we can explore interesting bias versus variance tradeoffs. Large k will yield high variance models and small k will yield high bias models.

We summarize the time and space complexity of learning and inference in mixtures of cutset networks in the following theorem.

Theorem 2. *The time complexity of learning mixtures of cutset networks is $O(n^2Nskt_{max})$ where s is the number of nodes in the cutset network, N is the number of examples, k is the number of mixture components, t_{max} is the maximum number of iterations for which EM is run and n is the number of variables. The space complexity of the algorithm is $O(nsk)$, which also bounds the space required by the mixtures of cutset networks. The time complexity of performing marginal and maximum-a-posteriori inference in a mixtures of cutset networks is $O(nks)$.*

Proof. As proved in Theorem 1, each CN in the ensemble requires $O(n^2Ns)$ time to be learned. There are k such CNs each learned once during an EM iteration on the weighted data. Hence the total time to learn a mixture is $O(n^2Nskt_{max})$. Each CN requires $O(ns)$ space and therefore the mixture of k CNs requires $O(nsk)$ space which also bounds the complexity for performing maginal and maximum-a-posteriori inference. □

3.7 Empirical Evaluation

The aim of our experimental evaluation is two fold: comparing the learning speed, measured in terms of CPU time, and accuracy, measured in terms of test set log likelihood scores, of our methods with state-of-the-art methods for learning tractable models.

3.7.1 Methodology and Setup

We evaluated our algorithms as well as the competition on 20 benchmark datasets shown in Table 3.1. The number of variables in the datasets ranged from 16 to 1556, and the number of training examples varied from 1.6K to 291K examples. All variables in our datasets are binary-valued for a fair comparison with other methods, who operate primarily on binary-valued input. These datasets or a subset of them have also been used by (Davis and Domingos, 2010; Rooshenas and

Lowd, 2014; Lowd and Rooshenas, 2013; Lowd and Davis, 2010; Gens and Domingos, 2013; Van Haaren and Davis, 2012).

We implemented three variations of our algorithms: (1) learning CNs without pruning (CN), (2) learning CNs with pruning (CNP) and (3) learning mixtures of CNs (MCNs). We compared their performance with the following learning algorithms from literature: learning sum-product networks with direct and indirect interactions (ID-SPN) (Rooshenas and Lowd, 2014), learning Markov networks using arithmetic circuits (ACMN) (Lowd and Rooshenas, 2013), learning mixture of trees (MT) (Meila and Jordan, 2000), Chow-Liu trees (Chow and Liu, 1968), learning Sum-Product Networks (LearnSPN) (Gens and Domingos, 2013) and learning latent tree models (LTM) (Choi et al., 2011). Most of the results on the datasets (except the results on learning Chow-Liu models) were made available to us by (Rooshenas and Lowd, 2014). They are part of the Libra toolkit available on Daniel Lowd's web page.

We smoothed all parameters using 1-laplace smoothing. For learning CNs without pruning, we stopped building the OR tree when the number of examples at the leaf node were fewer than 10 or the total entropy was smaller than 0.01. To learn MCNs, we varied the number of components from 5 to 40, in increments of 5 and ran the EM algorithm for 100 iterations or convergence whichever was earlier. For each iteration of EM, we could update both the structure and the parameters of the cutset network associated with each component. However, to speed up the learning algorithm, we chose to update just the parameters, utilizing the structure learned at the first iteration.

3.7.2 Learning Time

Table 3.1 shows the time taken by CN, CNP, MCN, ID-SPN and ACMN to learn a model from data. We gave a time limit of 48 hours to all algorithms and ran all our timing experiments on a quad-core Intel i7, 2.7 GHz machine with 8GB of RAM. The fastest cutset network learners, in order, are: CN, CNP, and MCN. On an average, ACMN is slower than MCN. ID-SPN is the slowest algorithm. In fact, ID-SPN did not finish on 8 out of the 20 datasets in 48 hours (note that

Table 3.1. Runtime Comparison (in seconds). Time-limit for each algorithm: 48 hours. †indicates that the algorithm did not terminate in 48 hours.

Dataset	Var#	Train	Valid	Test	CN	CNP	MCN	ID-SPN	ACMN
NLTCS	16	16181	2157	3236	0.2	0.4	36.5	307.0	242.4
MSNBC	17	291326	38843	58265	13.0	29.2	2177.7	90354.0	579.9
KDDCup2000	64	180092	19907	34955	95.9	197.8	1988.0	38223.0	645.5
Plants	69	17412	2321	3482	6.5	10.5	135.0	10590.0	119.4
Audio	100	15000	2000	3000	17.2	19.6	187.0	14231.0	1663.9
Jester	100	9000	1000	4116	14.0	11.8	101.2	†	3665.8
Netflix	100	15000	2000	3000	25.2	22.6	224.4	†	1837.4
Accidents	111	12758	1700	2551	15.7	22.1	195.4	†	793.4
Retail	135	22041	2938	4408	18.9	27.6	104.7	2116.0	12.5
Pumsb-star	163	12262	1635	2452	30.1	41.8	233.8	18219.0	374.0
DNA	180	1600	400	1186	13.8	6.9	57.7	150850.0	39.9
Kosarek	190	33375	4450	6675	65.9	102.5	141.2	†	585.4
MSWeb	294	29441	32750	5000	208.6	365.8	642.8	†	286.3
Book	500	8700	1159	1739	129.1	204.2	154.4	125480.0	3035.0
EachMovie	500	4524	1002	591	90.7	133.4	204.8	78982.0	9881.1
WebKB	839	2803	558	838	169.7	228.7	160.4	†	7098.3
Reuters-52	889	6532	1028	1540	397.1	650.4	1177.2	†	2709.6
20Newsgroup	910	11293	3764	3764	695.2	935.8	1525.2	†	16255.3
BBC	1058	1670	225	330	206.7	223.9	70.2	4157.0	1862.2
Ad	1556	2461	327	491	365.8	594.3	155.4	285324.0	6496.4

for the datasets on which ID-SPN did not finish in 48 hours, we report the test set log-likelihood scores from (Rooshenas and Lowd, 2014)). The best performing cutset network algorithm, MCN, was faster than ID-SPN on all 20 datasets and ACMN on 14 datasets. If we look at the learning time and accuracy (see Table 3.2) as a whole, CNP is the best performing algorithm, providing reasonably accurate results in quick time.

3.7.3 Accuracy

Table 3.2 shows the average test set log likelihood scores for the various benchmark networks while Table 3.3 shows head-to-head comparison of the six best performing algorithms namely CNP, MCN, ID-SPN, ACMN, MT and LearnSPN. Excluding the first two datasets where there

Table 3.2. Average test set log-likelihood. †indicates that the results of this algorithm are not available.

Dataset	CN	CNP	MCN	ID-SPN	ACMN	MT	Chow-Liu	LearnSPN	LTM
NLTCS	-6.10	-6.05	-6.00	-6.02	-6.00	-6.01	-6.76	-6.11	-6.49
MSNBC	-6.06	-6.05	-6.04	-6.04	-6.04	-6.07	-6.54	-6.11	-6.52
KDDCup2000	-2.21	-2.19	**-2.12**	-2.13	-2.17	-2.13	-2.32	-2.18	-2.18
Plants	-13.37	-13.25	-12.78	**-12.54**	-12.80	-12.95	-16.51	-12.98	-16.39
Audio	-46.84	-41.97	**-39.73**	-39.79	-40.32	-40.08	-44.35	-40.50	-41.90
Jester	-64.50	-55.26	**-52.57**	-52.86	-53.31	-53.08	-58.21	-53.48	-55.17
Netflix	-69.74	-58.72	**-56.32**	-56.36	-57.22	-56.74	-60.25	-57.33	-58.53
Accidents	-31.59	-30.66	-29.96	**-26.98**	-27.11	-29.63	-33.17	-30.04	-33.05
Retail	-11.12	-10.98	**-10.82**	-10.85	-10.88	-10.83	-11.02	-11.04	-10.92
Pumsb-star	-25.06	-24.28	-24.18	**-22.40**	-23.55	-23.71	-30.80	-24.78	-31.32
DNA	-109.79	-87.50	-85.82	-81.21	**-80.03**	-85.14	-87.70	-82.52	-87.60
Kosarek	-11.53	-11.07	**-10.58**	-10.60	-10.84	-10.62	-11.52	-10.99	-10.87
MSWeb	-10.20	-10.12	-9.79	**-9.73**	-9.77	-9.85	-10.35	-10.25	-10.21
Book	-40.19	-37.51	**-33.96**	-34.14	-35.56	-34.63	-37.84	-35.89	-34.22
EachMovie	-60.22	-57.71	**-51.39**	-51.51	-55.80	-54.60	-64.79	-52.49	†
WebKB	-171.95	-161.58	-153.22	**-151.84**	-159.13	-156.86	-164.89	-158.20	-156.84
Reuters-52	-91.35	-87.64	-86.11	**-83.35**	-90.23	-85.90	-96.85	-85.07	-91.23
20Newsgroup	-176.56	-161.68	**-151.29**	-151.47	-161.13	-154.24	-164.99	-155.93	-156.77
BBC	-300.33	-260.55	-250.58	**-248.93**	-257.10	-261.84	-261.41	-250.69	-255.76
Ad	-16.31	-16.14	-16.68	-19.00	-16.53	**-16.02**	-16.67	-19.73	†

are multiple winners, we can see that MCN has the best log-likelihood score on 9 out of the remaining 18 benchmarks, while ID-SPN is the second best performing algorithm, with the best log-likelihood score on 7 out of the 18 benchmarks. In the head-to-head comparison, MCN is better than CNP on 19 benchmarks, ID-SPN on 11 benchmarks, ACMN on 13 benchmarks, MT on 15 benchmarks and LearnSPN on 18 benchmarks. CNP is better than ID-SPN only on 1 benchmark, ACMN and MT on 2 benchmarks while it is better than LearnSPN on 6 benchmarks. A careful look at the datasets reveal that when the number of training examples is large, MCN and to some extent CNP are typically better than the competition. However, for small training set sizes, ID-SPN is the best performing algorithm. As expected, Chow-Liu trees and CNs are the worst-performing algorithms, the former underfits and the latter overfits.

MCN is consistently better than CNP which suggests that whenever possible *it is a good idea to use latent mixtures of simple models*. This conclusion can also be drawn from the performance of MT, which greatly improves the accuracy of tree distributions.

Table 3.3. Head-to-head comparison of the number of wins (in terms of average test set log-likelihood score) achieved by one algorithm (row) over another (column), for all pairs of the 6 best performing algorithms used in our experimental study.

	CNP	MCN	ID-SPN	ACMN	MT	LearnSPN
CNP	-	1	1	2	2	6
MCN	19	-	11	13	15	18
ID-SPN	19	8	-	16	16	20
ACMN	18	5	3	-	8	15
MT	18	5	3	12	-	16
LearnSPN	14	2	0	5	4	-

3.8 Chapter Summary

In this chapter, we presented cutset networks - a novel, simple and tractable probabilistic graphical model. At a high level, cutset networks are operational representation of Pearl's cutset condition-

ing method, with an OR tree modeling conditioning (at the top) and a tree distribution modeling inference over trees at the leaves. We developed an efficient algorithm for learning cutset networks from data. Our new algorithm uses a decision tree inspired learning algorithm for inducing the structure and parameters of the OR tree and the classic Chow-Liu algorithm for learning the tree distributions at the leaf nodes. We also presented an EM-based algorithm for learning mixtures of cutset networks.

Our detailed experimental study on a variety of benchmark datasets clearly demonstrated the power of cutset networks. In particular, our new algorithm that learns mixtures of cutset networks from data, was the best performing algorithm in terms of average test set log-likelihood score on 55% of the benchmarks when compared with 5 other state-of-the-art algorithms from literature. Moreover, our new *one-shot algorithm*, which builds a cutset network using the information gain heuristic and employs reduced-error pruning is not only fast (as expected) but also reasonably accurate on several benchmark datasets. This gives us a spectrum of algorithms for future investigations: fast, accurate one-shot algorithms and slow, highly accurate iterative algorithms based on EM.

CHAPTER 4

LEARNING ENSEMBLES OF CUTSET NETWORKS

4.1 Introduction

In Chapter 3, we showed that cutset networks (CNs) admit a polynomial-time learning algorithm when the number of nodes in the OR tree is bounded by a constant. A straight-forward approach is to generate all possible CNs having a small, fixed number of OR nodes and then select the one that has the highest log-likelihood score on the training data. Although tractable, the learning algorithm has high polynomial complexity and therefore we proposed a heuristic recursive algorithm to remedy the problem. Further, to avoid overfitting, we proposed a bottom-up post-pruning method in section 3.5.2. Recently, (Di Mauro et al., 2015) proposed a Bayesian learning approach to prevent overfitting, in lieu of the costly post-pruning step.

In this chapter, instead of using methods such as post-pruning and Bayesian learning to avoid overfitting, we propose to limit the depth of the OR tree, yielding a weak learner. We then improve the accuracy of weak learners using ensemble methods, namely we develop techniques that combine multiple CNs using popular ensemble learning approaches such as bagging and boosting.

We make the following contributions in this chapter. First, in section 4.2, we develop sequential boosting-based as well as parallel bagging-based algorithms for learning ensembles of CNs from data, leveraging vast amount of previous work on boosting and bagging algorithms (cf. (Zhou, 2012)) as well as their generalizations for density estimation (Rosset and Segal, 2002; Ridgeway, 2002; Welling et al., 2002). Second, in section 4.3, we perform and report on a comprehensive empirical evaluation, comparing our new algorithms with several state-of-the-art systems such as sum-product networks with direct and indirect interactions (Rooshenas and Lowd, 2014), latent tree models (Choi et al., 2011) and mixtures of cutset networks (Rahman et al., 2014) on a wide

variety of benchmark datasets. Our empirical evaluation clearly demonstrates the power of our new approaches; our new algorithms are better than competing systems on 12 out of 20 datasets in terms of average test set log-likelihood. This is significant because we compare with well-engineered, state-of-the-art systems.

The research in this chapter is based on (Rahman and Gogate, 2016a).

4.2 Ensembles of Cutset Networks

We define an ensemble of cutset networks (ECNs), denoted by f_M, as a collection of pairs $\{\langle \alpha_m, \mathcal{C}_m \rangle\}_{m=1}^{M}$ where $\mathcal{C}_m$ is a CN and α_m is its associated weight or coefficient such that $\alpha_m \geq 0$ and $\sum_{m=1}^{M} \alpha_m = 1$. f_M represents the following probability distribution.

$$f_M(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m \mathcal{C}_m(\boldsymbol{x})$$

where $\boldsymbol{x}$ is an assignment of values to all the variables in $\boldsymbol{X}$ in the domain and $\mathcal{C}_m(\boldsymbol{x})$ is the probability of $\boldsymbol{x}$ w.r.t. $\mathcal{C}_m$. We assume that each cutset network $\mathcal{C}_m$ in the ensemble is a member of some class $\mathcal{Q}^d$ of cutset networks, such that the depth of the OR tree of all networks in the class $\mathcal{Q}^d$ is bounded by d. Frequently, we will refer to the cutset networks as the *base models* of the ensemble f_M.

Given training data $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^N\}$, we can learn f_M by solving the following optimization problem:

$$f_M^* = \underset{f_M}{\operatorname{argmax}}\ \mathcal{L}(f_M; \boldsymbol{\mathcal{D}}) \tag{4.1}$$

where $\mathcal{L}(f_M; \boldsymbol{\mathcal{D}}) = \sum_{i=1}^{N} \log(f_M(\boldsymbol{x}^i))$ is the log-likelihood of $\boldsymbol{\mathcal{D}}$ given f_M. Since α_m's cannot be directly estimated from data (they are not observed), the optimization problem is multi-modal. Therefore, we seek approximate heuristic approaches to solve it. One such approach is to fix M and then use the EM algorithm (Dempster et al., 1977) to solve the optimization problem. This approach yields the MCN algorithm described in the previous chapter.

In this chapter, we propose to solve the optimization problem using sequential boosting-based approaches as well as parallel bootstrapping (bagging) approaches. Intuitively, our new algorithms are likely to yield more accurate models than the conventional EM algorithm which updates all base models and mixture weights *simultaneously* because they will have better convergence properties (Neal and Hinton, 1998), smaller computational complexity (since networks will be added sequentially or via bootstrapping), and superior ability to escape local maxima than the latter. As we will describe in the section on experiments, our experimental results clearly validate this intuition.

4.2.1 Boosting

In this subsection, we present three approaches for boosting CNs. The first approach is based on the boosting density estimation approach of (Rosset and Segal, 2002) (henceforth, called the BDE method); the second is based on a kernel-based generalization of the BDE method; and the third approach uses the sequential (or incremental) EM algorithm. We describe the three approaches in order next.

The BDE method sequentially grows the ensemble, adding a weak base learner at each stage. Formally, at each boosting stage m it seeks to find a weak learner $\mathcal{C}_m$ belonging to a class $\mathcal{Q}^d$ and a coefficient η_m to add to the current model f_{m-1} such that the log-likelihood of the new model $f_m = (1-\eta_m)f_{m-1} + \eta_m \mathcal{C}_m$, denoted by $\mathcal{L}(f_m; \boldsymbol{\mathcal{D}})$, is maximized where $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^N\}$ is a set of N training examples. Since optimizing the log-likelihood is hard (there is no closed form solution), the log-likelihood is approximated using the following expression (derived using Taylor series).

$$\mathcal{L}(f_m; \boldsymbol{\mathcal{D}}) \approx \mathcal{L}(f_{m-1}; \boldsymbol{\mathcal{D}}) + \frac{\eta_m}{1-\eta_m} \sum_{i=1}^{N} \frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)} \tag{4.2}$$

Assuming that η_m is a (small) constant, $\mathcal{L}(f_m; \boldsymbol{\mathcal{D}})$ can be maximized by maximizing $\sum_i \frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)}$. To maximize the latter, at each boosting step m, we search for a CN $\mathcal{C}_m \in \mathcal{Q}^d$ that maximizes the weighted log-likelihood of data, in which each example $\boldsymbol{x}^i$ is weighted by $\frac{1}{f_{m-1}(\boldsymbol{x}^i)}$. In other words,

examples which have lower probability according to the previous model receive higher weight and vice versa.

An algorithmic description of a scheme that uses the BDE method for boosting CNs is given in Algorithm 2. The algorithm begins by initializing the model f_0 to the uniform distribution. Then, at each iteration m, it updates the weight ω_m^i of each example $\boldsymbol{x}^i$ to $\frac{1}{f_{m-1}(\boldsymbol{x}^i)}$ (step 4), learns a new CN $\mathcal{C}_m$ that maximizes the weighted log-likelihood using the algorithm described in (Rahman et al., 2014; Di Mauro et al., 2015) (step 5), finds a weighting co-efficient η_m for the new model by performing a line search (step 6), and finally updates the model f_m with the new weighting co-efficient (step 7).

Algorithm 2: CN-Boosting($\boldsymbol{\mathcal{D}}, \boldsymbol{X}, M, d$)

Input : Dataset $\boldsymbol{\mathcal{D}}$, Variables $\boldsymbol{X}$, An integer M, maximum-depth d

Output: f_M

1 **begin**

2 $\quad f_0 =$ uniform distribution

3 $\quad$ **for** $m = 1$ *to* M **do**

4 $\quad\quad \omega_m^i = \frac{1}{f_{m-1}(\boldsymbol{x}^i)}\ \forall \boldsymbol{x}^i \in \boldsymbol{\mathcal{D}}$

5 $\quad\quad \mathcal{C}_m = \underset{\mathcal{C}}{\operatorname{argmax}} \sum_i \omega_m^i \mathcal{C}(\boldsymbol{x}^i)$ from $\mathcal{Q}^d$

6 $\quad\quad \eta_m = \underset{\eta}{\operatorname{argmax}} \sum_i \log((1-\eta) f_{m-1}(\boldsymbol{x}^i) + \eta \mathcal{C}_m(\boldsymbol{x}^i))$

7 $\quad\quad f_m = (1-\eta_m) f_{m-1} + \eta_m \mathcal{C}_m$

8 $\quad$ **end**

9 $\quad$ **return** f_M

10 **end**

Next, we derive an AdaBoost (Freund and Schapire, 1997) style algorithm for learning ECNs by generalizing the weight update rule in step 4 of Algorithm 2. To derive this rule, we rewrite the

weight as

$$\omega_{m+1}^i = \frac{1}{f_m(\boldsymbol{x}^i)} = \frac{1}{(1-\eta_m)f_{m-1}(\boldsymbol{x}^i) + \eta_m \mathcal{C}_m(\boldsymbol{x}^i)}$$

Dividing both the numerator and the denominator by $f_{m-1}(\boldsymbol{x}^i)$ and factoring out η_m we get:

$$\omega_{m+1}^i = \frac{\omega_m^i}{1 + \eta_m(\frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)} - 1)} \tag{4.3}$$

From (4.3), we can see that the ratio $\frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)}$ determines the relationship between ω_m^i and ω_{m+1}^i:

- **Case 1:** $\frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)} \geq 1 \Rightarrow \omega_{m+1}^i \leq \omega_m^i$: the weight of an example having higher probability in $\mathcal{C}_m$ than f_{m-1} is decreased.

- **Case 2:** $\frac{\mathcal{C}_m(\boldsymbol{x}^i)}{f_{m-1}(\boldsymbol{x}^i)} < 1 \Rightarrow \omega_{m+1}^i > \omega_m^i$: the weight of an example having higher probability in f_{m-1} than $\mathcal{C}_m$ is increased.

We can express the relationship between ω_{m+1}^i and ω_m^i, in a more general form using the following expression:

$$\omega_{m+1}^i = \frac{\omega_m^i}{1 + \beta_m K(\mathcal{C}_m(\boldsymbol{x}^i), f_{m-1}(\boldsymbol{x}^i), \epsilon)} \tag{4.4}$$

where K is a kernel-function for *smoothing* the (gradient) updates, $\beta_m \in (0, 1)$ is the step size and ϵ is a tolerance measure. For example, the following function can be used to smooth the updates:

$$K(\mathcal{C}_m(\boldsymbol{x}), f_{m-1}(\boldsymbol{x}), \epsilon) = \begin{cases} +1 & \text{if } (\frac{\mathcal{C}_m(\boldsymbol{x})}{f_{m-1}(\boldsymbol{x})} - 1) > \epsilon \\ -1 & \text{if } (1 - \frac{\mathcal{C}_m(\boldsymbol{x})}{f_{m-1}(\boldsymbol{x})}) > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Using (4.4) (with K given by (4.5)) instead of the rule given in step 4 of Algorithm 2 yields *smooth* updates in the following sense. In the BDE method, all weights change at each iteration by a different amount. When a smooth K such as the one given in (4.5) is used, a weight ω_{m+1}^i is updated iff the relative difference between $\mathcal{C}_m(\boldsymbol{x}^i)$ and $f_{m-1}(\boldsymbol{x}^i)$ for the corresponding example $\boldsymbol{x}^i$ is larger than the tolerance measure ϵ. Moreover, all weights (that are updated) are updated by the same amount, similar to AdaBoost (which updates all misclassified examples).

Our third approach uses the EM algorithm (Dempster et al., 1977) to solve the optimization problem at each iteration m. Note that in the approximation given in (4.2), η_m is assumed to be a (small) constant. In our EM-based approach similar to (Ridgeway, 2002), we remove this relaxation and jointly optimize η_m and $\mathcal{C}_m$, keeping f_{m-1} fixed.[1] The algorithm operates as follows. At each iteration m, the algorithm randomly guesses η_m and $\mathcal{C}_m$. It then alternates between the following expectation (E-Step) and maximization steps (M-Step) until convergence:

- E-Step : $\gamma_m^i = \frac{\eta_m \mathcal{C}_m(\boldsymbol{x}^i)}{(1-\eta_m) f_{m-1}(\boldsymbol{x}^i) + \eta_m \mathcal{C}_m(\boldsymbol{x}^i)}$
- M-step : $\begin{cases} \eta_m = \frac{1}{N} \sum_{i=1}^{N} \gamma_m^i \\ \mathcal{C}_m = \underset{\mathcal{C}}{\operatorname{argmax}} \sum_{i=1}^{N} \gamma_m^i \log(\mathcal{C}(\boldsymbol{x}^i)), \;\; \mathcal{C} \in \mathcal{Q}^d \end{cases}$

4.2.2 Bagging

Bootstrap aggregation (Bagging) is a method for combining several high variance models of the same kind trained on different subsets of the data (Breiman, 2001). The subsets are called *bootstrap samples* and are generated by sampling the original data with replacement. In various empirical studies, bagging has been shown to improve the accuracy of several supervised learning algorithms by reducing the variance of the learning algorithm when data is scarce. We use the following straight-forward extension of the general bagging method to learn ECNs.

Algorithm 3 presents the steps to learn a bagged ensemble of cutset networks. Each constituent CN $\mathcal{C}_m$ in the ensemble is learned by generating *bootstrap samples* from the data, and then maximizing the log-likelihood of the generated samples. The coefficients α_m's can be learned in several ways. One approach is to attach the same weight $1/M$ to each $\mathcal{C}_m$, yielding a simple average mixture. Another approach is to weigh each $\mathcal{C}_m$ by a value that is proportional to its likelihood. Here

[1]The difference between this EM-based method and the EM-based MCN algorithm proposed in Chapter 3 is that in the latter at each EM iteration, all cutset networks $\mathcal{C}_j$ and coefficients η_j for $j = 1$ to m are updated simultaneously while in the former only $\mathcal{C}_m$ and η_m are updated.

we use the latter approach because it often performs better than simple averaging in practice. Note that the negative log-likelihood measures the error of the base models and thus our one-shot weighing technique assigns high weight to low error models and vice versa. Since each bootstrap uses roughly 63.2% of the unique examples of $\boldsymbol{\mathcal{D}}$, this measure takes into account the out-of-bag sample error and therefore exempts us from using a validation set to estimate the coefficients.

Algorithm 3: CN-Bagging($\boldsymbol{\mathcal{D}}, \boldsymbol{X}, M, d$)

Input : Dataset $\boldsymbol{\mathcal{D}}$, Variables $\boldsymbol{X}$, An integer M, maximum depth d

Output: f_M

begin
  $f_M = \emptyset$
  for $m = 1$ *to* M **do**
    $\boldsymbol{\mathcal{D}}_m$ = booststrap N samples from $\boldsymbol{\mathcal{D}}$
    $\mathcal{C}_m = \underset{\mathcal{C}}{\operatorname{argmax}}\, \mathcal{L}(\mathcal{C}; \boldsymbol{\mathcal{D}}_m),\ \mathcal{C} \in \mathcal{Q}^d$
    $\alpha_m = \frac{1}{\mathcal{L}(\mathcal{C}_m; \boldsymbol{\mathcal{D}})}$
    $f_M = f_M \cup \langle \alpha_m, \mathcal{C}_m \rangle$
  end
  return f_M
end

To de-correlate the cutset networks in the bagged ensemble, we use the following *random forests* inspired approach (Breiman, 2001). At each node of the OR tree, we pick r variables uniformly at random and then use the splitting heuristic to select the best variable from these r variables. This randomization trick also reduces the time complexity of training a tree since at each splitting node we only consider a smaller subset of the available variables.

4.3 Experiments

We evaluated the performance of ECNs on 20 real world benchmark datasets described in Chapter 3.. The number of variables range from 16 to 1556 and the number of training examples vary from 1.6K to 291K examples. All variables are binary. We ran all our experiments on a quad-core Intel i7 2.7 GHz machine with 16GB RAM and ran each algorithm for 48 hours or until termination, whichever was earlier.

We use a different heuristic than maximum gain for choosing the next variable to condition on. We score each variable $u \in \boldsymbol{X}$ using $Score(u) = \sum_{v \in \boldsymbol{X} \setminus \{u\}} I(u, v)$ where $I(u, v)$ is the mutual information between u and v and choose a variable having the highest score. The rationale for this heuristic is that variables having a large score are likely to be involved in a large number of edges and thus likely to be a part of a cutset. In our experiments, we found that this heuristic is much superior to the one used in the previous chapter for learning CNs.

Our chosen base models are CNs and small MCNs with the number of components fixed to either 2 or 3 in an ensemble (chosen by the validation set). We also ensured that the total number of component Nets learned does not exceed 40. This was done to make fair comparisons with other tractable model learners. We introduced two types of randomizations in learning the base models: randomizing the pool of cutset variables at each splitting node, and randomizing the maximum depth of the OR tree. The former is equivalent to learning an ensemble of randomized cutset networks as done in random forests – at each splitting node we randomly sample without return 50% of the variables and apply our heuristic to choose the best one. The latter is similar to stacking – combining models of different complexities. At each iteration we randomly picked an integer depth $0 \leq \ell \leq \texttt{max depth}$ to learn the next base model. Here `max depth` is the maximum depth that a cutset network can have in an ensemble. Therefore, both boosting and bagging are performed using fixed depth as well as variable depth base models leading to four categories of algorithms: learning with fixed depth CNs, learning with variable depth CNs, learning

with fixed depth MCNs, and learning with variable depth MCNs. We used 1-laplace smoothing for the parameters.

4.3.1 Boosting Performance

We compare the following three different boosting techniques using CNs and MCNs as base models: the BDE method, our proposed modification to the BDE method (see Eq. (4.5)) which we call the generalized BDE (GBDE) and the sequential EM-algorithm which we refer to as SEQEM in brief. For the GBDE, we tried the following values for ϵ: $\{0.1,0.3,0.5,0.7,0.9\}$ and β_m: $\{0.01,0.05,0.2,0.4,0.6,0.8\}$. The best values were selected based on the accuracy attained on the validation set. In all three algorithms, we added a base model to the ensemble until the validation set likelihood decreased or the total number of boosting iterations reached 40. The maximum depth of CNs and MCNs was varied from 0 to 5 and the best depth was also chosen by the validation set. To learn simpler models than fully connected tree distributions at the leaves of the base models (and thus avoid overfitting), we removed all edges from the Chow-Liu trees whose mutual information was smaller than 0.005. For learning MCNs we generated bootstrap samples from the weighted datasets in both BDE and GBDE algorithms (Freund and Schapire, 1996). The parameters of the mixture were then optimized via the EM algorithm using the original training set for 50 iterations or until convergence.

Table 4.1 reports the average test set log-likelihood scores achieved by each algorithm in four different categories of algorithms. The last row reports the average score across all datasets for each algorithm and represents a quick shot statistic for comparing the performance of the various algorithms. SEQEM-boosting with fixed depth MCNs is the best performing algorithm scoring the highest average log-likelihood of -53.23. GBDE yields better generalization performance than the BDE in all four categories. Table 4.2 reports the learning time in seconds for the various methods. Variable depth boosted models are significantly faster to learn than fixed depth boosted models but has lower prediction accuracy than fixed depth models in most cases. Although boosted

ensembles with MCNs as base models always perform better than boosted ensembles with CNs as base models, they are much shower to learn because of the iterative nature of learning through EM. GBDE not only has better generalization performance than BDE, but also converges faster.

4.3.2 Bagging Performance

We varied the number of bags from 5 to 40 with increments of 5. The maximum depth of the OR trees was varied from 2 to 10. The number of bags and the depth was chosen based on the accuracy on the validation set. In bagging MCNs, we randomized the structure using bootstrap replicates and then learned the best parameters for that structure on the training set (Ammar et al., 2010). EM was run for 75 iterations or until convergence. Unlike boosting, randomization of the depth and the pool of cutset variables improved the accuracy and learning time in bagging. Table 4.3 shows the average test set log-likelihood and learning time of bagged ensembles of CNs while Table 4.4 shows the results of bagged ensembles of MCNs. From Table 4.3 we observe that bagged ensembles of variable depth CNs with randomized variable selection (VD_R) has the highest test set log-likelihood score of -54.09 averaged over the 20 datasets and the fastest learning time of 743.24 seconds. Similar phenomenon can be observed in bagged ensembles of MCNs in Table 4.4 – bag of variable depth MCNs has the highest test set log-likelihood score of -53.21 averaged over the 20 datasets with the fastest learning time of 1426.02 seconds. Varying the depth and randomizing the selection of cutset variables helps more in higher dimensional cases with very few samples (e.g. the last five datasets) while datasets with more samples benefit from only randomizing the selection of cutset variables. In general, increasing the variability of the base models improved the accuracy by reducing the variance as well as significantly scaled the learning.

4.3.3 Comparison with State-of-the-art

We also compared the accuracy and learning efficiency of ECNs to five other well-cited state-of-the-art tractable model learners: learning sum-product network with direct and indirect variable

Table 4.1. Test set log-likelihood scores of boosting algorithms. Winning scores are bolded and underlines highlight GBDE over BDE. FD:= Fixed Depth and VD:=Variable Depth.

Datasets	CN						MCN					
	FD			VD			FD			VD		
	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM
NLTCS	-6.03	-6.01	-6.01	-6.03	-6.02	-6.01	-6.00	-6.00	-6.00	-6.00	-6.02	-6.02
MSNBC	-6.26	-6.21	-6.15	-6.25	-6.23	-6.15	-6.21	-6.17	-6.25	-6.22	-6.23	-6.24
KDDCup2K	-2.19	-2.15	-2.15	-2.18	-2.17	-2.15	-2.14	-2.13	-2.13	-2.15	-2.14	-2.14
Plants	-13.03	-12.76	-12.72	-13.42	-13.11	-12.65	-12.44	-12.42	**-12.32**	-12.58	-12.69	-12.64
Audio	-40.59	-40.37	-39.94	-40.95	-40.67	-39.84	-39.80	-39.86	**-39.67**	-39.89	-40.08	-40.11
Jester	-53.25	-52.98	-52.87	-53.55	-53.45	-52.82	-52.57	-52.69	**-52.44**	-52.82	-52.94	-52.78
Netflix	-56.76	-56.73	-56.47	-57.62	-57.61	-56.44	-56.29	-56.39	**-56.13**	-56.47	-56.65	-56.65
Accidents	-30.09	-30.09	-29.45	-30.52	-30.42	-29.45	-29.41	-29.33	**-29.27**	-29.50	-29.67	-30.26
Retail	-10.93	-10.89	-10.81	-10.96	-10.88	-10.82	-10.83	-10.85	**-10.79**	-10.84	-10.83	-10.83
Pumsb-star	-24.08	-24.09	-23.45	-24.37	-24.25	-23.47	-23.43	-23.48	**-23.37**	-23.53	-23.80	-26.03
DNA	-86.24	-86.30	-86.12	-86.18	-85.82	-85.67	-85.00	-84.93	**-82.67**	-84.03	-84.68	-85.12
Kosarek	-11.03	-10.77	-10.62	-10.83	-10.78	-10.60	-10.57	-10.57	**-10.54**	-10.58	-10.59	-10.56
MSWeb	-10.04	-9.86	-9.73	-10.03	-9.89	-9.75	-9.85	-9.80	**-9.72**	-9.89	-9.83	-9.79
Book	-36.08	-35.91	-34.46	-36.02	-35.74	-34.48	-33.93	-33.85	-33.95	-33.99	-33.85	**-33.78**
EachMovie	-55.18	-53.46	-52.00	-54.61	-53.87	-51.53	-51.63	-51.75	**-51.14**	-51.75	-51.48	-51.92
WebKB	-156.46	-155.08	-152.86	-154.95	-155.10	-152.53	-151.64	-151.45	-151.60	-151.51	**-150.71**	-150.84
Reuters-52	-85.82	-84.90	-84.03	-85.16	-84.83	-83.69	-83.65	-83.61	**-82.29**	-84.09	-83.73	-82.65
20Newsgrp.	-156.16	-155.61	-153.57	-155.85	-155.77	-153.12	-153.52	-152.90	**-151.75**	-152.59	-152.89	-153.17
BBC	-247.01	-247.44	-251.96	-250.92	-249.53	-251.81	-244.61	**-237.87**	-237.94	-242.43	-238.59	-248.32
Ad	-15.74	-15.90	-14.37	-15.75	-16.09	**-14.36**	-14.48	-14.97	-14.58	-14.65	-14.65	-14.50
Average LL	-55.15	-54.87	-54.49	-55.31	-55.11	-54.37	-53.90	-53.67	**-53.23**	-53.78	-53.60	-54.22

Table 4.2. Learning time comparison of boosting methods. FD:= Fixed Depth and VD:=Variable Depth.

Datasets	CN						MCN					
	FD			VD			FD			VD		
	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM
NLTCS	27.6	23.7	181.2	6.5	8.8	697.5	512.8	63.6	193.0	431.2	36.3	59.8
MSNBC	366.3	245.9	14962.2	136.0	67.7	4915.3	4224.6	5496.7	106.4	817.5	793.0	84.4
KDDCup2K	600.7	1064.9	5785.3	1113.8	470.1	1291.8	15454.7	2645.0	1731.5	1948.7	2313.5	3463.2
Plants	973.2	861.5	2773.7	81.3	95.3	3177.4	2494.6	1367.7	2340.0	2294.3	391.7	412.3
Audio	1384.6	909.0	2889.4	157.6	116.6	4593.8	2744.5	2417.9	2865.4	2630.4	721.6	1286.9
Jester	1540.0	1118.3	2152.6	211.7	71.4	2141.0	1695.2	1419.5	6437.1	1231.0	129.0	648.4
Netflix	5610.4	990.7	4569.4	101.1	115.0	3001.9	3328.4	1895.3	5321.3	2501.2	444.4	1273.1
Accidents	2049.2	814.0	3385.9	71.0	70.3	4849.3	3484.3	2184.5	7228.4	2774.8	354.7	731.7
Retail	213.4	72.1	7641.1	139.7	34.9	6070.4	4416.3	814.1	4065.4	3779.4	649.9	792.4
Pumsb_Star	1055.0	280.7	4419.9	153.7	65.1	4079.7	4173.1	2674.8	3864.3	2712.9	423.0	58.9
DNA	89.7	123.8	421.3	24.6	28.3	470.9	446.6	218.3	153.2	528.8	80.9	163.6
Kosarek	967.9	915.7	13352.1	473.3	365.4	4466.3	9474.6	2612.5	10891.2	8893.8	568.3	4323.6
MSWeb	1309.7	1858.6	17431.3	453.3	448.6	17421.8	13207.5	4159.4	16136.6	12167.8	4591.4	6370.0
Book	411.2	337.4	8756.6	827.5	1020.2	8655.2	6650.1	1442.8	5451.1	4063.4	3316.0	6182.6
EachMovie	3053.3	672.4	4037.3	684.9	996.4	3622.6	4646.3	3436.4	2073.3	1754.7	1432.9	679.3
WebKB	493.1	1079.0	2537.5	2010.0	452.3	1982.2	4754.5	1769.1	2038.2	2827.6	3078.2	3170.5
Reuters-52	5309.4	1497.4	10680.9	2042.5	727.4	8898.6	14995.2	3413.0	5268.6	6318.6	1067.7	5838.8
20NewsGrp.	12950.5	7458.0	25961.2	5147.0	4661.7	23137.8	27310.0	14772.9	10045.2	14157.6	13074.6	32987.2
BBC	2353.1	1209.0	1190.5	547.0	214.8	947.0	3774.4	1663.2	1697.8	3022.8	1305.9	694.4
Ad	2314.9	442.9	8800.2	372.0	451.3	7269.8	7834.9	2043.9	3041.0	2838.6	1425.9	3485.6
Average Time	2153.7	1098.8	7096.5	737.7	524.1	5584.5	6781.1	2825.5	4547.5	3884.8	1810.0	3635.3

Table 4.3. Average test set log-likelihood scores and learning time (in seconds) of bagged ensembles of cutset networks (CNs). FD:= Fixed Depth CNs (without randomized variable selection), FD_R:= Fixed Depth CNs with randomized variable selection, VD:= Variable Depth CNs (without randomized variable selection) and VD_R:= Variable Depth CNs with randomized variable selection. Bold values indicate the best values achieved for the dataset.

Datasets	FD		FD_R		VD		VD_R	
	LL	Time	LL	Time	LL	Time	LL	Time
NLTCS	-6.02	7.10	**-6.00**	2.77	-6.03	3.78	-6.03	2.46
MSNBC	-6.08	1037.46	-6.08	834.41	-6.17	242.98	-6.18	421.88
KDD2KCup	-2.16	4036.43	-2.15	1292.48	-2.16	2203.90	-2.15	991.36
Plants	-12.43	202.97	**-12.35**	43.68	-12.56	175.74	-12.50	39.13
Audio	-40.33	181.58	**-40.16**	66.83	-40.58	143.51	-40.44	71.07
Jester	-53.13	97.48	**-52.91**	41.73	-53.23	87.49	-53.17	39.43
Netflix	-57.11	214.56	**-56.57**	82.91	-57.34	172.67	-57.01	55.82
Accidents	-30.40	407.52	**-29.98**	62.11	-30.30	468.07	-30.26	68.12
Retail	-10.94	549.00	-10.89	249.16	-10.91	391.60	-10.89	228.94
Pumsb_star	-24.22	427.80	**-24.04**	127.68	-24.40	325.18	-24.34	133.20
DNA	-85.58	50.90	-84.21	22.01	-82.30	59.96	**-82.18**	18.73
Kosarek	-10.90	1882.70	**-10.73**	597.79	-10.82	2355.44	-10.75	621.16
MSWeb	-9.85	6199.35	**-9.79**	2780.89	-9.88	4465.47	-9.84	1158.97
Book	-35.97	1800.05	-36.05	1948.28	**-35.75**	3382.05	-35.77	1425.51
EachMovie	-54.29	904.19	-53.79	357.56	-53.43	1026.41	**-53.37**	468.71
WebKB	-155.59	1140.33	-155.12	632.09	-154.42	1056.76	**-154.15**	739.36
Reuters	-86.11	2388.42	-84.53	1690.28	-85.49	5265.21	**-84.25**	1914.25
20NewsGr.	-156.21	7929.19	-154.75	3903.13	-155.06	7368.10	**-154.35**	3953.20
BBC	-243.56	1301.06	-244.56	538.71	-239.46	1275.44	**-238.43**	630.48
Ad	-15.89	1947.93	-15.64	3499.17	**-15.62**	2201.61	-15.68	1883.04
Average	-54.84	1635.30	-54.52	938.68	-54.30	1633.57	**-54.09**	**743.24**

interactions (ID-SPN) (Rooshenas and Lowd, 2014), learning Markov networks using arithmetic circuits (ACMN) (Lowd and Rooshenas, 2013), learning mixtures of trees(MT) (Meila and Jordan, 2000), learning sum-product networks (SPN) (Poon and Domingos, 2011) and learning latent tree models (LTM) (Choi et al., 2011). Table 4.5 reports the performance of these algorithms. These results were taken from Chapter 3. For bagging and boosting we are only reporting results for settings selected using the validation set for a fair comparison. Comparing the time for Bagging from Table 4.6 with that of Boosting, we see that bagging is the fastest algorithm among the ensemble learning techniques for cutset networks. Unlike boosting, larger depth trees and randomization

Table 4.4. Average test set log-likelihood scores and learning time (in seconds) of bagged ensembles of small mixtures of cutset networks (MCNs). FD:= Fixed Depth MCNs (without randomized variable selection), FD_R:= Fixed Depth MCNs with randomized variable selection, VD:= Variable Depth MCNs (without randomized variable selection) and VD_R:= Variable Depth MCNs with randomized variable selection. Bold values indicate the best values achieved for the dataset.

Datasets	FD		FD_R		VD		VD_R	
	LL	Time	LL	Time	LL	Time	LL	Time
NLTCS	-6.01	21.75	**-6.00**	9.28	-6.02	18.84	-6.01	15.25
MSNBC	-6.08	2158.86	**-6.06**	4849.93	-6.11	1019.29	-6.10	779.88
KDD2KCup	-2.14	19635.64	-2.13	7725.90	-2.14	13373.30	-2.13	3058.92
Plants	-12.38	256.71	**-12.22**	670.70	-12.34	248.28	-12.30	128.79
Audio	-39.95	348.01	**-39.72**	279.86	-40.01	312.79	-39.91	273.84
Jester	-52.69	171.41	**-52.53**	143.94	-52.71	143.27	-52.71	120.55
Netflix	-56.81	443.25	**-56.27**	255.46	-56.76	316.84	-56.46	216.70
Accidents	-29.72	554.36	**-29.42**	208.41	-29.71	543.19	-29.58	203.71
Retail	-10.88	607.02	-10.84	312.43	-10.85	665.81	**-10.83**	422.11
Pumsb_star	-23.67	525.32	**-23.51**	331.86	-23.85	418.08	-23.69	146.92
DNA	-85.62	39.57	-84.15	36.65	-81.93	58.34	**-81.53**	31.96
Kosarek	-10.65	2268.80	-10.57	1166.24	-10.60	2902.81	**-10.56**	2502.24
MSWeb	-9.78	10325.48	**-9.72**	7264.64	-9.77	7072.79	-9.74	5745.33
Book	-34.36	1923.58	-34.31	1103.91	**-33.85**	1447.24	-33.89	1913.05
EachMovie	-52.18	747.58	-51.80	462.82	-51.51	678.21	**-51.47**	519.58
WebKB	-152.47	1159.21	-151.82	943.96	-151.02	1162.07	**-150.95**	861.10
Reuters	-84.47	2736.97	-83.31	4882.07	-83.81	6696.68	**-83.02**	2103.03
20NewsGr.	-152.75	5337.42	-151.62	3918.66	-151.90	7391.86	**-151.41**	5469.69
BBC	-242.37	1480.25	-240.26	844.19	-238.17	1684.16	**-236.99**	992.65
Ad	-15.12	2898.29	-15.02	2349.46	-14.94	2270.32	**-14.91**	3015.16
Average	-54.00	2681.97	-53.56	1888.02	-53.40	2421.21	**-53.21**	**1426.02**

significantly improves the accuracy of the model. Table 4.5 and Table 4.6 essentially compares the time versus accuracy trade-offs of boosting and bagging with CNs. ECNs algorithms are clearly the best performing algorithms outperforming the competition on 12 out of the 20 datasets with 1 tie. ECNs are almost always better than MCNs. This shows that our new bootstrap and sequential optimization approaches are superior to the non-sequential EM algorithm used in MCNs.

4.4 Chapter Summary

In this chapter, we presented novel boosting and bagging algorithms for learning ensembles of cutset networks from data. We performed a comprehensive empirical evaluation comparing our algorithms to state-of-the-art algorithms as well as to each other. Our results clearly show that our new additive models are quite powerful and superior to state-of-the-art algorithms.

Table 4.5. Test set log-likelihood comparison. (Ties are not bolded).

Dataset	#Var	#Train	#Valid	#Test	ECN		MCN	ID-SPN	ACMN	MT	SPN	LTM
					Bag	Boost						
NLTCS	16	16181	2157	3236	-6.00	-6.00	-6.00	-6.02	-6.00	-6.01	-6.11	-6.49
MSNBC	17	291326	38843	58265	-6.06	-6.15	-6.04	-6.04	-6.04	-6.07	-6.11	-6.52
KDDCup2K	64	180092	19907	34955	-2.13	-2.13	**-2.12**	-2.13	-2.17	-2.13	-2.18	-2.18
Plants	69	17412	2321	3482	**-12.22**	-12.32	-12.74	-12.54	-12.80	-12.95	-12.98	-16.39
Audio	100	15000	2000	3000	-39.72	**-39.67**	-39.73	-39.79	-40.32	-40.08	-40.50	-41.90
Jester	100	9000	1000	4116	-52.53	**-52.44**	-52.57	-52.86	-53.31	-53.08	-53.48	-55.17
Netflix	100	15000	2000	3000	-56.27	**-56.13**	-56.32	-56.36	-57.22	-56.74	-57.33	-58.53
Accidents	111	12758	1700	2551	-29.42	-29.27	-29.96	**-26.98**	-27.11	-29.63	-30.04	-33.05
Retail	135	22041	2938	4408	-10.83	**-10.79**	-10.82	-10.85	-10.88	-10.83	-11.04	-10.92
Pumsb-star	163	12262	1635	2452	-23.51	-23.37	-24.18	**-22.40**	-23.55	-23.71	-24.78	-31.32
DNA	180	1600	400	1186	-81.53	-82.67	-85.82	-81.21	**-80.03**	-85.14	-82.52	-87.60
Kosarek	190	33375	4450	6675	-10.56	**-10.54**	-10.58	-10.60	-10.84	-10.62	-10.99	-10.87
MSWeb	294	29441	32750	5000	-9.72	-9.72	-9.79	-9.73	-9.77	-9.85	-10.25	-10.21
Book	500	8700	1159	1739	-33.85	**-33.78**	-33.96	-34.14	-36.56	-34.63	-35.89	-34.22
EachMovie	500	4524	1002	591	-51.47	**-51.14**	-51.39	-51.51	-55.80	-54.60	-52.49	†
WebKB	839	2803	558	838	-150.95	**-150.71**	-153.22	-151.84	-159.13	-156.86	-158.20	-156.84
Reuters-52	889	6532	1028	1540	-83.02	**-82.29**	-86.11	-83.35	-90.23	-85.90	-85.07	-91.23
20Newsgrp.	910	11293	3764	3764	-151.41	-151.75	**-151.29**	-151.47	-161.13	-154.24	-155.93	-156.77
BBC	1058	1670	225	330	**-236.99**	-237.87	-250.58	-248.93	-257.10	-261.84	-250.69	-255.76
Ad	1556	2461	327	491	-14.91	**-14.36**	-16.68	-19.00	-16.53	-16.02	-19.73	†
Average LL					-53.16	-53.16	-54.50	-53.89	-55.83	-55.55	-55.32	†

Table 4.6. Runtime comparison (in seconds). †indicates that the algorithm did not terminate in 48 hrs.

Datasets	Boosting						Bagging		MCN	ID-SPN	ACMN
	BDE		GBDE		SEQEM						
	CN	MCN	CN	MCN	CN	MCN	CN	MCN			
NLTCS	27.60	512.80	82.60	289.51	181.17	193.04	2.77	9.28	36.57	307.00	242.40
MSNBC	366.28	4224.64	992.60	5787.68	14962.22	106.38	834.41	4849.93	2177.73	90354.00	579.90
KDDCup2K	600.73	15454.69	6966.55	13703.42	5785.30	1731.47	991.36	3058.92	1988.49	38223.00	645.50
Plants	973.17	2494.60	2668.30	1829.74	2773.67	2340.01	43.68	670.70	135.00	10590.00	119.40
Audio	1384.60	2744.51	1529.60	2045.94	2889.43	2865.37	66.83	279.86	187.78	14231.00	1663.90
Jester	1539.95	1695.21	3354.30	1063.90	2152.58	6437.08	41.73	143.94	101.15	†	3665.80
Netflix	5610.40	3328.44	3982.86	2243.79	4569.44	5321.28	82.91	255.46	224.38	†	1837.40
Accidents	2049.23	3484.30	1888.71	2334.04	3385.86	7228.44	62.11	208.41	195.49	†	793.40
Retail	213.41	4416.32	285.11	1132.00	7641.05	4065.44	228.94	422.11	104.67	2116.00	12.50
Pumsb-star	1055.03	4173.11	1015.13	2908.71	4419.91	3864.32	127.68	331.86	233.79	18219.00	374.00
DNA	89.69	446.61	128.02	331.83	421.34	153.18	18.73	31.96	57.69	150850.00	39.90
Kosarek	967.93	9474.57	2564.82	5693.66	13352.09	10891.22	597.79	2502.24	141.16	†	585.40
MSWeb	1309.71	13207.53	3646.54	12260.44	17431.33	16136.63	2780.89	7264.64	642.80	†	286.30
Book	411.21	6650.12	4451.52	1484.70	8756.64	5451.08	3382.05	1447.24	154.42	125480.00	3035.00
EachMovie	3053.31	4646.34	2973.99	3560.84	4037.28	2073.33	468.71	519.58	204.81	78982.00	9881.10
WebKB	493.07	4754.49	1185.25	1643.65	2537.48	2038.20	739.36	861.10	160.40	†	7098.30
Reuters-52	5309.41	14995.21	5770.73	4637.31	10680.87	5268.56	1914.25	2103.03	1525.20	†	2709.60
20Newsgrp.	12950.45	27310.00	7463.05	7774.85	25961.20	10045.16	3953.20	5469.69	1177.16	†	16255.30
BBC	2353.06	3774.40	1497.82	1823.51	1190.47	1697.80	630.48	992.65	70.21	4157.00	1862.20
Ad	2314.91	7834.92	263.50	3188.44	8800.22	3040.97	2201.61	3015.16	155.38	†	6496.40
Average Time	2153.66	6781.14	2635.55	3786.90	7096.48	4547.45	958.47	1721.89	**483.71**	†	2909.19

CHAPTER 5

MERGING STRATEGIES FOR SUM-PRODUCT-CUTSET NETWORKS

5.1 Introduction

In this chapter, we propose sum-product-cutset networks (SPCNs) – a novel representation that combines the power of deep, tractable probabilistic architectures called sum-product networks (SPNs) (Poon and Domingos, 2011) with fast learnability of cutset networks (CNs). Recall that an SPN consists of alternating levels of sum and product nodes with trivial univariate distributions at each leaf. Sum nodes are latent and represent mixtures over their child sub-SPNs while product nodes represent decomposition (partition) of observed variables into independent components. Our empirical evaluations in previous chapters show that the performance of CNs can be greatly improved using mixtures or sum nodes. We hypothesize that by using product nodes which take advantage of decomposition, in addition to the sum nodes, we can further improve their performance. This yields a novel deep architecture, which we call SPCNs, which has a sum-product network at the top with cutset networks at the leaves.

SPCNs have two major advantages over SPNs and CNs. First, SPCNs allow conditioning (sum nodes) over both observed and latent variables. Models learned using this general assumption are often much more compact and accurate than SPNs and CNs. Second, unlike SPNs, SPCNs have CNs as leaf nodes. CNs are much superior in terms of representation power than univariate distributions; they represent a joint probability distribution over large number of variables, yet admit polynomial time inference and learning algorithms.

The second contribution of this chapter is proposing an algorithm that learns more compact and accurate representations of SPCNs. Existing structure learning algorithms for SPNs, and the ones proposed for CNs in the previous chapters (and thus by extensions for SPCNs) induce a tree

structured model – there exists a single conditioning path from the root to each internal node. Tree SPCNs are a very small and inefficient sub-class of the possible structures, and thus by inducing only tree models our learning algorithms may miss important structures having high accuracy. We address this limitation by developing post-processing approaches that induce graph SPCNs from tree SPCNs. The main idea in our approach is as follows. We first learn a tree structured model over latent and observed nodes using standard algorithms, and then convert the tree structured model to a graph structured model by processing the learned model in a bottom-up fashion, merging two sub-networks (sub-SPNs, sub-CNs or sub-SPCNs) if the distributions represented by them are similar and defined over the same variables. To convert this idea into a general-purpose algorithm, we have to solve two problems: (1) how to find similar sub-networks, and (2) how to merge them into one sub-network. Both problems are computationally expensive to solve and therefore we develop approximate algorithms for solving them, which is the main contribution of this chapter. The key benefits of graph representations over tree representations include (1) smaller computational complexity which facilitates faster online inference, and (2) better generalization accuracy because of reduced variance, at the cost of slight increase in the learning time.

The third contribution of this chapter is a thorough experimental evaluation of our proposed merging algorithms on twenty benchmark datasets, all of which were used in several previous studies. Our experiments clearly show that merging always improves the performance of tree networks, measured in terms of test set log-likelihood score and prediction time. We also experimentally compared bagged ensembles of graph structured models with state-of-the-art approaches such as ensembles of cutset networks (Rahman and Gogate, 2016a) introduced in Chapter 4, sum-product networks with direct and indirect interactions (Rooshenas and Lowd, 2014), sum-product networks learned via the SVD-based approach (Adel et al., 2015), arithmetic circuits with Markov networks (Lowd and Rooshenas, 2013), and mixtures of cutset networks (Rahman et al., 2014) on the same datasets, and found that our new approach yields better test set log likelihood score on 8 out of the 20 datasets with two ties. This clearly demonstrates the power of our new merging algorithms.

The rest of the chapter is organized as follows. In the next section, we revisit SPNs and describe a top-down algorithm for learning these models from data. In section 5.3, we formally introduce SPCNs. In section 5.4, we describe powerful merging approaches for converting an arbitrary tree structured representation to a graph structured representation. Experimental results are presented in section 5.5 followed by a summary of our achievements in section 5.6.

The research presented in this chapter is based on (Rahman and Gogate, 2016b).

5.2 Top Down Learning of SPNs

As mentioned in section 2.3.5, any (discrete) probability distribution over a set of variables $\boldsymbol{X}$ can be expressed using an arithmetic circuit (AC) (Darwiche, 2003) or a sum-product network (SPN) (Poon and Domingos, 2011). SPNs used in this dissertation are equivalent to ACs (as well as AND/OR decision diagrams (Mateescu et al., 2008)) defined over latent and observed variables. However, in order to be consistent, we will use the term SPNs throughout the dissertation. We will distinguish between the following types of SPNs: SPNs with sum nodes defined over only the observed variables, SPNs with sum nodes defined over only latent variables and SPNs with sum nodes defined over either observed or latent variables. All three have different representation powers. SPNs with observed sum nodes are CNs with product nodes; SPNs with both observed and latent sum nodes are our proposed SPCNs and are more general and therefore more powerful than the other representations.

The literature abounds with algorithms for learning the structure of SPNs and ACs from data, starting with the work of Lowd and Domingos (Lowd and Domingos, 2008) who proposed to learn ACs over observed variables by using the AC size as a learning (inductive) bias within a Bayesian network structure learning algorithm, and then compiling the induced Bayesian network to an AC. Later Lowd and Rooshenas (Lowd and Rooshenas, 2013) extended this algorithm to learn a Markov network having small AC size. The latter performs much better in terms of test set

log likelihood score than the former because of the increased flexibility afforded by the undirected Markov network structure.

A limitation of the two aforementioned approaches for learning ACs is that they do not use latent variables; it turns out that their accuracy can be greatly improved using latent variables. Unfortunately, the parameter learning problem (a sub-step in structure learning) – the problem of learning the weights or probabilities of a given SPN structure – is much harder in presence of latent variables. In particular, the optimization problem is non-convex, which necessitates the use of algorithms such as gradient descent and expectation maximization that only converge to a local minima. However, since learning is often an offline process, this increase in complexity is often not a big issue.

The first approach for learning the structure of SPNs with latent variables as sum nodes and observed variables appearing only at the leaf nodes is due to Gens and Domingos (Gens and Domingos, 2013). An issue with this approach is that it learns only directed trees instead of (directed acyclic) graphs and as a result is unable to fully exploit the power and flexibility of SPNs. To address this limitation, (Dennis and Ventura, 2012; Peharz et al., 2013; Dennis and Ventura, 2015) proposed to learn graph structured SPNs over latent variables while (Rooshenas and Lowd, 2014) proposed to learn graph structured SPNs over observed variables at the leaves. A drawback of these approaches is that they are unable to learn a graph SPN over both observed and latent variables. In this chapter, we address this limitation.

In the following, we focus on top-down approaches that directly learn the structure of SPNs from data. Instead of learning Bayesian and Markov networks and then compiling them into SPNs (this is the approach used in (Lowd and Domingos, 2008; Lowd and Rooshenas, 2013)), the key advantage of this direct approach is that the size of the SPN can be controlled in a straight-forward manner, which is typically bounded from above by the data size.

Algorithm 4 shows a generic recursive learning algorithm for learning tree SPNs from data, which is loosely based on the algorithm proposed by Gens and Domingos (Gens and Domingos,

Algorithm 4: LEARNSPN($\boldsymbol{\mathcal{D}}$,$\boldsymbol{X}$)

Input: Set of Training Instances $\boldsymbol{\mathcal{D}}$ and set of variables $\boldsymbol{X}$
Output: An SPN representing a distribution over $\boldsymbol{X}$

begin
  // 1. Base Case
  if *conditions for inducing the base models are satisfied*
    then return LEARNBASEMODEL($\boldsymbol{\mathcal{D}}$,$\boldsymbol{X}$)
  // 2. Decomposition Step
  if $\boldsymbol{X}$ *can be partitioned into subsets* $\boldsymbol{X_j}$
    then return $\prod_j$ LEARNSPN($\boldsymbol{\mathcal{D}}$,$\boldsymbol{X_j}$)
  // 3. Splitting Step
  Partition $\boldsymbol{\mathcal{D}}$ into subsets of similar instances $\boldsymbol{\mathcal{D}}_i$
  return $\sum_i \frac{|\boldsymbol{\mathcal{D}}_i|}{|\boldsymbol{\mathcal{D}}|}$ LEARNSPN($\boldsymbol{\mathcal{D}}_i$,$\boldsymbol{X}$)
end

2013). The algorithm has three steps: base case, decomposition and splitting. In the base case, if the conditions for learning the base model are satisfied, for example, when the size of the training data is small or only one variable remains, then the algorithm learns the corresponding trivial distribution and terminates the recursion. In the decomposition step, the algorithm tries to partition the variables into roughly independent components $\boldsymbol{X_j} \subseteq \boldsymbol{X}$ such that $P(\boldsymbol{X}) = \prod_j P(\boldsymbol{X_j})$ and recurses on each component, inducing a product node. If neither the base case nor the conditions for the decomposition step are satisfied, then the algorithm partitions the training instances into clusters of multiple instances, inducing a sum node, and recurses on each part.

Several techniques proposed in literature for learning SPNs can be understood as special cases of Algorithm 4, with the difference between them being the approaches used at the three steps. Table 5.1 gives examples of techniques from the SPN literature that are based on Algorithm 4. Gens and Domingos (Gens and Domingos, 2013), and Gogate et al. (Gogate et al., 2010) stop when only one variable remains and induce a univariate distribution; Rahman et al. (Rahman et al., 2014), Rahman and Gogate (Rahman and Gogate, 2016a) and Vergari et al. (Vergari et al., 2015), stop when the entropy of the data is small or use a Bayesian criteria, and induce an SPN corresponding to a tree distribution (tree BN/MN) at the leaves using the Chow-Liu algorithm (Chow and Liu, 1968) in polynomial time. Rooshenas and Lowd (Rooshenas and Lowd, 2014)

Table 5.1. Examples of SPN structure learning approaches in the literature that follow the prescription given in Algorithm 4. Base case is the stopping criteria for the recursive algorithm.

Reference	Base Case	Decomposition	Splitting
(Gens and Domingos, 2013)	Univariate distribution	Independence tests	Latent Variables
(Gogate et al., 2010)	Univariate distribution	Independence assumption	Conjunctive fixed-length features
(Rahman et al., 2014)	Tree Markov networks	not used	Observed variables
(Rahman and Gogate, 2016a)	Tree Markov networks	not used	Observed and Latent variables
(Vergari et al., 2015)	Tree Markov networks	Independence tests	Latent Variables
(Rooshenas and Lowd, 2014)	Arithmetic Circuits	Independence tests	Latent Variables

learns an SPN over observed variables in the base case using the algorithm described in (Lowd and Domingos, 2008). In the decomposition step, Gens and Domingos, Rooshenas and Lowd, and Vergari et al. use pair-wise variable independence tests (e.g., the G-test) for inducing the product nodes; Gogate et al. uses no independence tests and instead assume that each split decomposes the variables into multiple components; while Rahman et al. ignore the decomposition step inducing only sum nodes. Gens and Domingos, Rooshenas and Lowd, and Vergari et al. split only over latent variables using a naive Bayes mixture model with hard EM, Gogate et al. and Rahman et al. split only over observed variables which are heuristically chosen or their features, while Rahman and Gogate split over both latent and observed variables.

Although, the structure learning problem is NP-hard in SPNs having only observed variables as well as in SPNs having both observed and latent variables, the parameter (weight) learning problem is easier in the former than the latter. In particular, parameter learning can be done in closed form when the SPN has only observed variables. On the other hand, the optimization problem is non-convex in the presence of latent variables and one has to use iterative algorithms having high computational complexity such as hard and soft EM to solve the non-convex problem (cf. (Poon and Domingos, 2011; Meila and Jordan, 2000; Rahman and Gogate, 2016a)). Thus, although latent variables help yield a more powerful representation, they often significantly increase the learning time.

5.3 Sum-Product-Cutset Networks

Formally, a sum-product-cutset network (SPCN) is a rooted directed acyclic graph which consists of three types of internal nodes: latent-sum nodes, observed-sum nodes, and product nodes with cutset networks as leaf nodes (see for example Figure 5.1). Latent-sum nodes represent mixtures, and observed-sum nodes represent conditioning of observed variables, similar to the nodes in the OR search tree. Similar to SPNs, we impose the following restrictions:

1. Each node t represents a conditional distribution over variables mentioned in its descendants given an assignment of values to all observed and latent variables on all paths between the root node and t.

2. The distributions over child nodes of a latent sum-node are defined over the same set of variables.

3. Each observed sum-node is labeled by a discrete variable v and the distribution over its child nodes is defined over the remaining discrete (namely, all but v) variables.

4. Let a product node t have k child nodes. Then for each pair of children $\langle ch_i, ch_j \rangle$ of t $Scope(ch_i) \cap Scope(ch_j) = \emptyset$.

Representationally, SPCNs are equivalent to SPNs in that we can easily transform an SPCN to an SPN in linear time. The algorithm for doing this is straight-forward. All we have to do is transform each tree Bayesian network to an SPN by converting the former to a junction tree and then inducing an SPN from the junction tree. In summary,

Proposition 1. Each SPCN having $O(p)$ parameters can be converted in linear time and space to an equivalent SPN having $O(p)$ parameters.

Thus, inference procedures for SPNs can be used to perform inference over SPCNs. However, note that learning algorithms for the two are much different. In fact, learnng is much faster and easier in SPCNs and as a result SPCNs are often more accurate than SPNs.

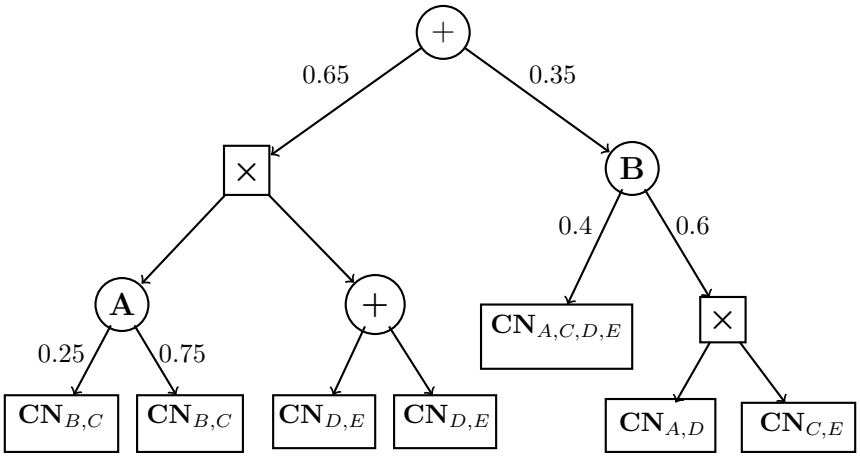


Figure 5.1. An SPCN rooted by a latent-sum node (denoted by a $+$) over discrete variables $\boldsymbol{X} = \{A, B, C, D, E\}$. All observed and latent variables are over binary domains and left edges represent conditioning by value 0 or false and right edges represent conditioning by value 1 or true. A and B are the only observed cutset variables. Product nodes (denoted by $\times$) decompose the model differently and each leaf node (denoted by a rectangle) is a cutset network (CN) over the remaining observed variables (appears in the subscript as a comma separated list).

5.4 Merging Strategies: From Trees to Graphs

A key problem with existing methods for learning SPCNs is that they induce tree models, except at the leaves. It is well known in the probabilistic inference literature (Dechter and Mateescu, 2007; Darwiche, 2003, 2001) that tree SPCNs can be exponentially larger than graph SPCNs, which are obtained from the former by merging identical sub-SPCNs (see Fig.5.2(b) and (c)). Thus, converting tree SPCNs to graph SPCNs is a good idea because they can significantly improve the time required to make predictions.

From a learning point of view, graph SPCNs can potentially improve the generalization performance by addressing the following issue associated with the LEARNSPN algorithm: as the depth of the node increases,[1] the number of training examples available for learning a sub-SPCN rooted at the node decreases exponentially. Merging increases the number of examples available at a node, since examples from all directed paths from the root to the node can be combined. This reduces the variance of the parameter estimates while having no effect on their bias. Since the

[1]The depth of a node equals the number of sum nodes from the root to the node.

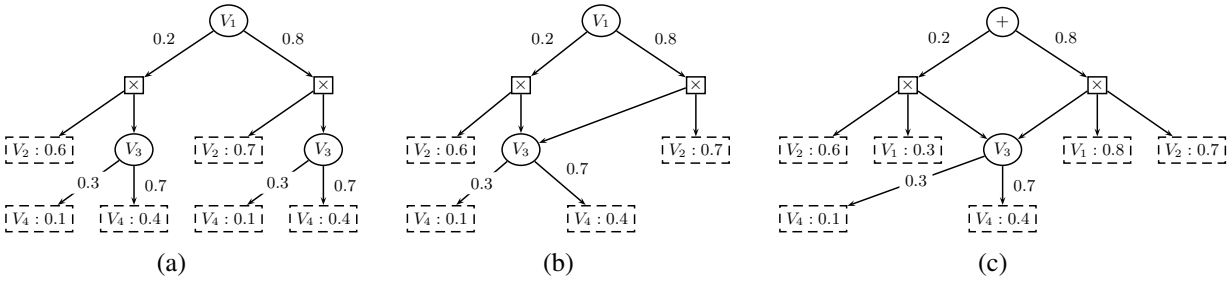


Figure 5.2. Three example SPNs over variables $\{V_1, V_2, V_3, V_4\}$. We are assuming that all variables are binary and take values from the domain $\{0, 1\}$. Leaf nodes express univariate distributions. For example, the node $V_2 : 0.6$ expresses the probability distribution $P(V_2 = 1) = 0.6$. Sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. (a) Tree SPN (SPN which is a rooted directed acyclic tree) that decomposes according to a tree Markov network $V_4 - V_3 - V_1 - V_2$. (b) Graph SPN that is equivalent to the tree SPN given in (a) obtained by merging identical sub-trees. (c) Graph SPCN over latent and observed sum nodes.

mean-squared error of the model equals bias squared plus the variance, graph SPCNs are likely to be more accurate than tree SPCNs. The following proposition formalizes this intuition:

Proposition 2. Let S_1, S_2 and $S_{1,2}$ be three (sub-)SPCNs having the same structure and defined over the same variables but whose parameters are estimated from training examples T_1, T_2 and $T_{1,2} = T_1 \cup T_2$ respectively. Then assuming that the datasets are generated uniformly at random from a distribution whose structure decomposes according to S_1 (and thus S_2 and $S_{1,2}$), the sample variance of $S_{1,2}$ is smaller than S_1 and S_2.

Proof. The sample variance of S_1, S_2 and $S_{1,2}$ is given by $Var(S_1)/|T_1|$, $Var(S_2)/|T_2|$ and $Var(S_{1,2})/|T_1 \cup T_2|$ respectively where $Var(S_1)$, $Var(S_2)$ and $Var(S_{1,2})$ is the (population) variance of the distributions induced by S_1, S_2 and $S_{1,2}$. Since $Var(S_1) = Var(S_2) = Var(S_{1,2})$ (our assumption), $|T_1 \cup T_2| \geq |T_1|$ and $|T_1 \cup T_2| \geq |T_2|$, the proof follows. □

5.4.1 Our Approach

The main idea in our approach is to relax the identical sub-SPCN requirement and merge *similar* sub-SPCNs. We use this relaxation because the sub-SPCNs are estimated from data and the likeli-

Algorithm 5: MERGE(S,$\boldsymbol{X}$,ϵ)

Input: SPCN S
Output: Merged SPCN S'

```
begin
    S' = S
    repeat
        for i = 1 to |X| do
            S_i = sub-SPCNs in S' having exactly i variables in their scope
            ρ = Partition S_i into cells having identical scopes
            for each cell ρ_j of ρ do
                Merge all sub-SPCNs in ρ_j such that the distance between them is bounded
                by ε and S' is a DAG
    until convergence;
    return S'
```

hood that they will be identical is slim to none. In this context, we develop methods for answering the following two questions: which sub-SPCNs to merge and how to merge them.

One approach for selecting candidate sub-SPCNs for merging is to compare the distance between the distributions represented by the two sub-SPCNs, given that they are defined over the same variables, and check if the distance is smaller than a threshold. However, computing the distance between two sub-SPCNs can be quite time-consuming. Therefore, we propose to use the following mean-field style approximation (Wiegerinck, 2000) of the distance between the two distributions:

$$D(P||Q) \approx \frac{1}{|\mathbf{X}|} \sum_{v \in \mathbf{X}} D(P(v)||Q(v))$$

where P and Q are two distributions over $\mathbf{X}$ and D is a distance function (e.g., KL divergence, relative error, Hellinger distance, etc.). Since single-variable marginal distributions in each sub-SPCN can be computed in time that is linear in the number of nodes of the sub-SPCN (and in practice can be pre-computed), our proposed distance method is also linear time.

Next, we describe our greedy, bottom-up approach for merging similar sub-SPCNs of a given SPCN S (see Algorithm 5). The algorithm begins by initializing S' to S and repeats the following steps until convergence. For all sub-SPCNs S_i of S' that are defined over exactly i variables, it

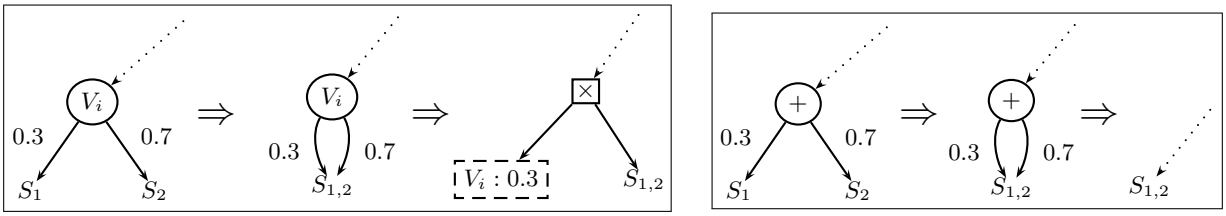
partitions the sub-SPCNs based on their scopes such that all sub-SPCNs having the same scope are in the same cell (part) ρ_j of the partition ρ. Then, in each cell ρ_j, ensuring that S' remains a DAG, it merges all sub-SPCNs such that the distance between them is bounded by ϵ, a user-defined constant that can be set using a validation set. Another option is to merge two sub-SPCNs if the accuracy on the validation set improves, thereby using a greedy strategy (in our experiments, we used both strategies). Note that the for-loop of the algorithm operates in a bottom-up fashion similar to reduced-error pruning in decision trees. The loop starts at the leaves, which are sub-SPCNs having just one variable in their scope ($i = 1$), and then proceeds towards the root which includes all variables in its scope ($i = |\boldsymbol{X}|$). The algorithm is guaranteed to converge in finite number of iterations because at each iteration, the size of the SPCN can only decrease or remain the same.

5.4.2 Practical Merging Strategies

We complete the description of the algorithm by describing how to merge two similar sub-SPCNs S_1 and S_2. A straight-forward method is to merge the datasets at the two sub-SPCNs and then learn a new graph sub-SPCN, say $S_{1,2}$ from the new dataset. An issue with this approach is that since our basic algorithm (see Algorithm 4) learns tree SPCNs, we have to call Algorithm 5 again to convert the newly created tree SPCN to a graph SPCN. This may yield a self-recursive algorithm with infinite loops that may not terminate. To overcome this computational difficulty, we propose to not relearn the structure, but only update the weights. In particular, we use the following approach. We consider two candidate structures for the merged sub-SPCN; the first structure is identical to S_1 and the second to S_2. Then, we learn the weights of the two candidate sub-SPCN using the merged dataset and choose the one that yields the maximum improvement in accuracy (log-likelihood score) over the validation set.

There are two types of merging that require special attention. The first type is when the two sub-SPCNs are children of the same sum node. In this case, if the sum node corresponds to splitting

over an observed variable, we can replace the sum-node by a product node having two children as depicted in Fig. 5.3(a). On the other hand, if the sum node is a latent node then the sum node can be deleted without changing the underlying distribution. This is depicted in Fig. 5.3(b). This type of merging is useful because it substantially simplifies the model, allowing us to either prune sub-SPCNs (see Fig. 5.3(b)) or take advantage of problem decomposition (see Fig. 5.3(a)). This yields better generalization and faster inference.



(a) Merge child nodes of observed sum node (b) Merge child nodes of latent sum node

Figure 5.3. Figure demonstrating how to simplify and thus reduce the size of the SPCN after merging. As before, sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. $S_{1,2}$ is an SPCN obtained by merging SPCNs S_1 and S_2. (a): shows how the SPCN can be reduced when the two child nodes of an observed sum node are merged. The node $V_i : 0.3$ represents a univariate probability distribution over V_i with $P(V_i = 1) = 0.3$. (b): shows how the SPCN can be reduced when the two child nodes of a latent sum node are merged.

A second type of merging that requires special attention is when the two sub-SPCNs to be merged correspond to tree Markov (or Bayesian) networks over the observed variables. In this case, unlike in the general case, we propose to learn both the structure and parameters of the merged sub-SPCN (using the merged dataset). This is because both the structure and parameter learning problem in such SPCNs can be solved in polynomial time using the Chow-Liu algorithm (Chow and Liu, 1968).

5.5 Experiments

5.5.1 Setup

We evaluated the impact of merging on 20 real world benchmark datasets (see Table 3.1) which has been used in the previous chapters as well. All of our experiments were performed on a quad-core Intel i7 2.7 GHz machines with 16 GB RAM. Each algorithm was given a time bound of 48 hours, after which the algorithm was terminated.

5.5.2 Algorithms Evaluated

We implemented two variants of SPCNs: SPCNs in which sum nodes split over value assignments to a latent variable and SPCNs in which sum nodes split over value assignments to a heuristically chosen observed variable. Henceforth, we will call the two SPCNs L-SPCNs and O-SPCNs respectively. We learned tree versions of both SPCNs using Algorithm 4. We used tree Markov networks (MNs) as base models in both SPCNs; as mentioned earlier tree MNs can be learned in polynomial time using the Chow-Liu algorithm.

To learn sum nodes in L-SPCNs, following Gens and Domingos (Gens and Domingos, 2013), we employed hard EM over a naive Bayes mixture model with three random restarts for 15 iterations to split the training instances into two clusters, i.e. we only considered binary splits for latent sum nodes for better regularization and faster learning as in (Vergari et al., 2015). To learn sum nodes in O-SPCNs, we employed two heuristics proposed in our previous work (Rahman et al., 2014; Rahman and Gogate, 2016a). The first heuristic selects an observed variable that has the highest information gain. The second heuristic selects an observed variable based on the following mutual information based criteria: given a set of variables $\boldsymbol{X}$ and training data $\boldsymbol{\mathcal{D}}$, we score each variable $v \in \boldsymbol{X}$ using $Score(v) = \sum_{u \in \boldsymbol{X} \setminus v} I_{\boldsymbol{\mathcal{D}}}(v, u)$ where $I_{\boldsymbol{\mathcal{D}}}(v, u)$ is the mutual information between v and u according to $\boldsymbol{\mathcal{D}}$ and choose a variable having the highest score. Variables having high mutual information score are likely to yield better decompositions, which in turn will likely yield small depth SPCNs having high generalization accuracy.

In both L-SPCNs and O-SPCNs, we learn product nodes using the technique described in Gens and Domingos (Gens and Domingos, 2013). We first compute the mutual information graph given data (similar to the Chow-Liu algorithm). This graph is a complete weighted graph over all variables, in which each edge is weighted by the mutual information between the two corresponding variables. Then, we prune weak edges from the graph using a threshold chosen from β: $\{0.001, 0.0015, 0.01, 0.5\}$. Finally, we find connected components of the pruned graph, and recursively learn a sub-SPCNs over variables and data in each connected component.

We varied the depth h of SPCNs from $\{4, 5, 6, 7, 10\}$.[2] We use the following stopping criteria for learning the base model (tree Markov network): stop when the number of samples N at a node is less than 10 or the maximum depth is reached. All parameters in the model were smoothed using 1-Laplace smoothing.

For each possible configuration of h and β we learned both a tree L-SPCN and a tree O-SPCN. In case of O-SPCNs, we also varied the heuristic to choose an observed variable. The best tree SPCN in each category was chosen according to the average log-likelihood score achieved on the validation set and provided as the input to the merging algorithm (see Algorithm 5). Then, we applied practical merging and simplification strategies described in Section 5.4.2 on the merged SPCN and report the test set log-likelihood score of the merged model that achieved the highest log-likelihood score on validation set.[3] We used Manhattan distance to measure the distance between two candidate sub-SPCNs and chose a threshold (ϵ) from $\{0.0001, 0.001, 0.01, 0.1\}$ using the validation set. Finally, after each merge we performed the following sanity/model complexity check. If the merged sub-SPCN had smaller log-likelihood than a tree Markov network on the validation set, we replaced the merged sub-SPCN by the latter.

[2]Note that the overall depth of the SPCN is h plus the depth of the SPCN corresponding to the tree Markov network (our base model). Thus, the overall depth can be quite large (> 30 in most cases).

[3]Our experiments showed that merging sub-SPCNs that are rooted at child nodes of the same sum node (the cases given in Fig. 5.3) was often more beneficial as compared to merging sub-SPCNs that are child nodes of two different sum nodes.

5.5.3 Impact of Merging on Test Set Log-Likelihood

Table 5.2 shows the results of our experiments for evaluating the impact of merging on the accuracy, time complexity and prediction time of L-SPCNs and O-SPCNs. In terms of learning time, we see that for L-SPCNs, merging requires a significant amount of time. This is to be expected because parameter learning is computationally expensive in presence of latent variables. To update the parameters of candidate sub-, we ran hard EM with the merged dataset for 20 iterations or until convergence. For some L-SPCNs (e.g. Plants), merging was a factor of 200 slower than learning tree models. The reason for this anomaly is that the corresponding tree L-SPCNs have large number of latent sum nodes. On the other hand, merging is significantly faster in O-SPCNs than L-SPCNs because the parameters are updated in closed-form, by making only one pass over the data as well as the model.

We measure the prediction time by the number of edges attached to the sum nodes (see columns |T|, |G| and CR in Table 5.2) since the prediction time is linearly proportional in the number of these weights. We see that in general merging yields reductions in complexity of inference by reducing the size of the network in majority of cases.

In terms of accuracy, we see from Table 5.2 that merging improves the test set log-likelihood score for the majority of datasets, clearly demonstrating our intuition that it will yield better generalization, primarily because it significantly reduces the variance at the cost of slightly increasing the bias.

5.5.4 Comparison with State-Of-The-Art

Finally, we demonstrate that we can achieve state-of-the-art performance using our merging algorithm. For this, following our previous work (Rahman and Gogate, 2016a; Vergari et al., 2015), we learn bagged ensemble of tree SPCNs and graph SPCNs. It was shown in previous studies that bagged ensembles of tree SPCNs (especially L-SPCNs) achieves state-of-the-art results. In

Table 5.2. Table showing the impact of merging on the average test-set log likelihood, time complexity and prediction time of L-SPCNs and O-SPCNs (all values rounded to two decimal places). We use the following notation: (1) T-LL: Average test-set log likelihood for the tree SPCNs; (2) G-LL: average test-set log likelihood for the graph SPCNs obtained from the tree SPCNs by merging similar sub-SPCNs; (3) |T|: number of parameters in the tree SPCN; (4) |G|: number of parameters in the graph SPCN; (5) CR:=Compression Ratio = $\frac{|T|}{|G|}$; (6) T-Time: Tree SPCN learning time in seconds and (7) G-Time: Time in seconds required by the merging algorithm (thus the total learning time for graph SPCN is T-time+G-time seconds). In each row, **bold** values indicate the best score for each of the two SPCN categories: L-SPCN and O-SPCN.

Datasets	L-SPCN							O-SPCN						
	T-LL	G-LL	|T|	|G|	CR	T-time	G-time	T-LL	G-LL	|T|	|G|	CR	T-time	G-time
NLTCS	-6.03	-6.04	5498	3988	1.38	5.37	396.69	-6.04	-6.05	1406	1152	1.22	0.98	4.69
MSNBC	-6.46	-6.46	2780	2440	1.14	109.38	53.49	-6.09	**-6.08**	20032	9478	2.11	6.36	1245.62
KDD	-2.14	-2.14	11516	6670	1.73	199.13	15119.05	-2.22	**-2.19**	34328	16608	2.07	91.38	59.54
Plants	-12.80	**-12.69**	65132	47802	1.36	68.44	17775.76	-13.83	**-13.49**	86530	36960	2.34	9.56	14.12
Audio	-40.11	**-40.02**	12798	10804	1.18	68.30	1995.94	-42.06	-42.06	6142	6142	1.00	10.74	3.90
Jester	-53.12	**-52.97**	12798	10002	1.28	39.09	20.89	-55.38	**-55.36**	6142	4996	1.23	6.51	2.39
Netflix	-56.71	**-56.64**	12798	11604	1.10	62.64	2287.78	-58.64	-58.64	6142	6142	1.00	19.95	2.35
Accidents	-30.09	**-30.01**	14206	13322	1.07	58.23	2089.49	-30.83	-30.83	6846	6846	1.00	14.13	3.90
Retail	-10.88	**-10.87**	3238	2162	1.50	51.15	75.25	-11.02	**-10.95**	6302	3158	2.00	32.69	15.06
Pumsb_star	-24.17	**-24.10**	19558	17604	1.11	66.05	2314.47	-24.42	**-24.34**	20222	18338	1.10	20.6	14.93
DNA	-85.90	**-85.51**	5758	4320	1.33	8.26	11.26	-90.43	**-87.49**	11262	1430	7.88	3.76	9.48
Kosarek	-10.62	-10.62	5318	5318	1.00	219.01	200.11	-11.10	**-10.98**	11902	6712	1.77	79.55	46.66
MSWeb	-9.95	**-9.90**	32926	16484	2.00	490.12	29482.04	-10.07	**-10.06**	15086	12770	1.18	209.54	21.07
Book	-34.80	**-34.76**	15998	11998	1.33	220.56	129.98	-38.60	**-37.44**	31740	11916	2.66	387.75	10.75
EachMovie	-52.07	-52.07	15998	15998	1.00	94.92	91.31	-59.99	**-58.05**	31745	19846	1.60	176.95	6.18
WebKB	-154.86	**-153.55**	26846	20134	1.33	157.89	78.64	-172.08	**-161.17**	53438	10046	5.32	287.01	249.27
Reuters-52	-84.70	**-83.90**	56894	46232	1.23	478.65	1331.38	-90.43	**-87.49**	56638	28334	2.0	485.6	428.4
20NewsGrp.	-154.35	-154.67	58238	43684	1.33	913.81	3457.07	-163.35	**-161.46**	57982	29016	2.0	827.71	705.53
BBC	-256.05	**-253.45**	33854	21160	1.60	98.55	53.93	-272.98	**-260.59**	63242	8454	7.48	163.47	142.89
Ad	-16.77	-16.77	49790	49790	1.00	244.44	155.53	-17.37	**-15.39**	62098	31070	2.00	953.70	832.40

our evaluation, we wanted to see whether we would be able to match or exceed these results using bagged ensemble of graph SPCNs. As a strong baseline, we also compare with five other state-of-the-art tractable model learners: (1) learning sum-product networks with direct and indirect variable interactions (ID-SPN) (Rooshenas and Lowd, 2014), learning Markov networks using arithmetic circuits (ACMN) (Lowd and Rooshenas, 2013), learning mixtures of cutset networks (MCN) (Rahman et al., 2014), learning sum-product networks via SVD based algorithm (SPN-SVD) (Adel et al., 2015) and learning ensembles of cutset networks (ECN) (Rahman and Gogate, 2016a). We report the results of IDSPN and ACMN from (Rooshenas and Lowd, 2014), MCN from (Rahman et al., 2014), SPN-SVD from (Adel et al., 2015) and ECN from (Rahman and Gogate, 2016a).

In our experiments, we fixed the number of bags to 40 following (Rahman and Gogate, 2016a). Instead of performing a grid search, we performed random search (Bergstra and Bengio, 2012) to create a configuration for the models in the ensemble. Each component model was then weighted according to its likelihood on the training set. To get better accuracy, we treated the bagged ensemble of L-SPCNs and O-SPCNs as an SPCN having one latent sum node as the root and each independent component (bag) as its child sub-SPCN. The benefit of this approach is that instead of optimizing the local log-likelihood scores of individual SPCNs, while merging, we can directly optimize the global log-likelihood.

Table 5.3 shows the bagged ensemble scores of L-SPCNs and O-SPCNs before and after merging as well as the best log-likelihood score obtained to date using the competing approaches mentioned above. Bagged graph SPCNs, especially L-SPCNs, performed significantly better than the state-of-the-art on all of the high dimensional datasets with very competitive scores on the others. This suggests that merging is especially useful for accurately modeling relationships in high-dimensional data (see also Table 5.2).

Table 5.3. Average test set log-likelihood comparison with state-of-the-art tractable model learners. **Bold** values indicate the winning score for the corresponding dataset. T-LL: Bagged LL of tree SPCNs and G-LL: Bagged LL of graph SPCNs. Column "Best-LL to date" gives the best log-likelihood score to date for each dataset obtained using the following competing approaches: ID-SPN (Rooshenas and Lowd, 2014), ACMN (Lowd and Rooshenas, 2013), MCN (Rahman et al., 2014), SPN-SVD (Adel et al., 2015), and ECN (Rahman and Gogate, 2016a).

Datasets	|Var|	|Train|	|Valid|	|Test|	L-SPCN		O-SPCN		Best-LL to date
					T-LL	G-LL	T-LL	G-LL	
NLTCS	16	16181	2157	3236	-6.01	-6.00	-6.01	-6.00	-6.00
MSNBC	17	291326	38843	58265	-6.45	-6.39	-6.10	-6.10	**-6.04**
KDD	64	180092	19907	34955	-2.13	-2.12	-2.14	-2.13	-2.12
Plants	69	17412	2321	3482	-12.31	-12.03	-12.25	-12.21	**-11.99**
Audio	100	15000	2000	3000	-39.57	**-39.49**	-40.35	-40.31	-39.67
Jester	100	9000	1000	4116	-52.65	-52.47	-53.56	-53.13	**-41.11**
Netflix	100	15000	2000	3000	-55.92	**-55.84**	-56.69	-56.65	-56.13
Accidents	111	12758	1700	2551	-29.41	-29.32	-29.81	-29.82	**-24.87**
Retail	135	22041	2938	4408	-10.85	-10.82	-10.87	-10.85	**-10.60**
Pumsb_star	163	12262	1635	2452	-23.82	-23.67	-23.85	-23.81	**-22.40**
DNA	180	1600	400	1186	-86.63	-80.89	-85.97	-84.79	**-80.03**
Kosarek	190	33375	4450	6675	-10.71	-10.55	-10.85	-10.74	**-10.54**
MSWeb	294	29441	32750	5000	-9.84	-9.78	-9.77	-9.76	**-9.22**
Book	500	8700	1159	1739	-36.49	-34.25	-36.35	-35.89	**-30.18**
EachMovie	500	4524	1002	591	-54.70	**-50.72**	-55.82	-53.07	-51.14
WebKB	839	2803	558	838	-170.27	**-150.04**	-166.65	-152.82	-150.10
Reuters-52	889	6532	1028	1540	-84.32	**-80.66**	-86.00	-82.66	-82.10
20NewsGrp.	910	11293	3764	3764	-151.48	**-150.80**	-158.40	-154.28	-151.29
BBC	1058	1670	225	330	-265.89	**-233.26**	-244.12	-238.61	-236.82
Ad	1556	2461	327	491	-16.33	-14.58	-15.69	**-14.34**	-14.36

5.6 Chapter Summary

In this chapter, we proposed sum-product-cutset networks (SPCNs) by combining the features of sum-product networks and cutset networks to scale up the learning and accuracy of a deep probabilistic model. We presented a novel algorithm for learning graph structured SPCNs from tree structured SPCNs by merging similar sub-SPCNs. Our proposed algorithm for finding and merging similar sub-SPCNs is general enough to serve as a template for incorporating suitable functions that measure similarity between sub-SPCNs as well as for performing arbitrary mergings. Our

experimental evaluation clearly shows that graph SPCNs can significantly boost the accuracy and prediction time of tree SPCNs by substantially reducing the number of parameters that the learning algorithm needs to induce from data. We also investigated the merit of learning ensembles of graph SPCNs, building on our previous work on learning ensembles of tree SPCNs, for a variety of high dimensional real world datasets, and comparing them to other state-of-the-art tractable model learners. Our experimental results showed that ensembles of graph SPCNs significantly outperform the state-of-the-art learners, clearly demonstrating the efficacy of our proposed approach.

CHAPTER 6

LEARNING HYBRID SUM-PRODUCT-CUTSET NETWORKS

6.1 Introduction

Cutset networks (CNs) and sum-product-cutset networks (SPCNs) have been introduced as tractable PGMs for representing joint distributions over discrete variables. Most real-world application domains have a combination of both discrete and continuous variables. For example, in predicting whether a patient has heart disease, several measurements like cholesterol, resting blood pressure, max heart rate etc. are continuous valued while the gender of the patient and the type of chest pain (Angina, Abnormal Angina, Nonanginal, Asymptomatic) are discrete valued measures. The most common approach for handling continuous variables when learning PGMs is to discretize them (Pernkopf and Bilmes, 2005; Friedman and Goldszmidt, 1996; Kozlov and Koller, 1997) which results in loss of information (Yang and Webb, 2009). On the other hand (Silva et al., 2011), (Uria et al., 2013), and (Tang et al., 2012) proposed to learn models only over continuous variables removing the discrete variables present in the domain.

In this chapter, we propose hybrid sum-product-cutset networks (HSPCNs) – a simple and natural extension to SPCNs to handle mixed domains and develop novel learning algorithms for them. HSPCNs yield a rich class of models that generalize several related, previously proposed tractable models such as hybrid tree-augmented Bayesian networks (Friedman et al., 1997), and hybrid Bayesian multinets (Geiger and Heckerman, 1996a). They also include hybrid cutset networks as well as hybrid sum-product networks (SPNs) as special cases. HSPCNs allow conditioning (sum nodes) over both discrete observed and latent variables and have tree structured conditional linear Gaussian networks (CLGs) (Lauritzen, 1996; Lauritzen and Wermuth, 1984, 1989; Heckerman

and Geiger, 1995; Bøttcher, 2004; Murphy, 1998) as leaf nodes. Conditional linear Gaussian networks are Bayesian networks which represent joint distributions over both discrete and continuous variables with the following restrictions – discrete variables cannot have continuous parents and continuous distributions are assumed to be Gaussians. Tree CLG networks are a subclass of CLG networks which allow variables to have at most one parent. Thus they are similar to the discrete tree distributions discussed in Section 2.3.1 over mixed domains with aforementioned constraints. Tree CLG networks are much superior in terms of representation power than univariate distributions as in SPNs; they represent a joint probability distribution over large number of discrete and continuous variables, yet admit polynomial time inference and learning algorithms. Most real world application domains contain both discrete and continuous variables, and HSPCNs enable an application designer to develop rich *density estimators* in such domains.

The rest of the chapter is organized as follows. In Section 6.2, we present a brief background on conditional linear Gaussian networks and formalize the notation used throughout the chapter. Section 6.3 introduces our proposed hybrid architectures and we describe a learning algorithm for it in section 6.4. In section 6.5, we present results of our detailed experimental evaluation, in which we compare HSPCNs with four generatively learned models: cutset networks, naive Bayes (baseline), Bayesian multi-nets and SPNs, on eighteen datasets from the UCI machine learning repository. In our evaluation, we considered two tasks: density estimation and classification, with the performance over the two tasks measured using test set log-likelihood score and classification accuracy respectively. We found that HSPCNs are substantially more accurate than the competition on both tasks. We conclude by summarizing our contributions in Section 6.6.

6.2 Background

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_m\}$ denote the set of discrete and continuous variables respectively. Let $\{\Delta_1, \ldots, \Delta_n\}$ be the set of domains of the discrete variables where Δ_i is the domain of X_i which denotes the values that the random variable X_i can take. Let $\mathbf{x} = (x_1, \ldots, x_n)$

and $\mathbf{y} = (y_1, \ldots, y_m)$ denote an assignment of values to all discrete and continuous variables respectively where $x_i \in \Delta_i$ and $y_j \in \mathbb{R}$. We assume that each continuous variable Y_i is normally distributed with mean μ_i and variance σ_i^2, namely $P(y_i) = \mathcal{N}(y_i; \mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp(-\frac{(y_i-\mu_i)^2}{2\sigma_i^2})$.

A tree CLG is a pair $\langle \mathcal{G}, \mathbf{\Theta} \rangle$ where $\mathcal{G} = \langle \mathbf{X} \cup \mathbf{Y}, \mathbf{E} \rangle$ is a directed acyclic tree in which nodes represent random variables in $\mathbf{X} \cup \mathbf{Y}$ and edges represent direct probabilistic dependencies among them, and $\mathbf{\Theta}$ is a set of functions, one for each variable, representing the conditional probability distribution (CPD) of that variable given its parents in $\mathcal{G}$. The CPDs and $\mathcal{G}$ have the following restrictions:

1. Each node has at most one parent.

2. A discrete variable has no continuous parents.

3. The CPD of a continuous variable is either a table of Gaussians or a linear Gaussian. If a continuous variable Y_i has a discrete parent X_j, then the CPD of Y_i is a table of normal distributions, one for each value of the discrete parent. Formally, let $\Delta_j = \{x_{j,1}, \ldots, x_{j,d}\}$ be the domain of X_j, then $P(Y_i = y_i | X_j = x_{j,k}) = \mathcal{N}(y_i; \mu_{i|k}, \sigma_{i|k}^2)$ where $\mu_{i|k}$ and $\sigma_{i|k}^2$ are conditional mean and variance respectively. On the other hand, if Y_i has a continuous parent Y_j, then the conditional mean of Y_i is linear in $Y_j = y_j$ as $\mu_{i|j} = \mu_i + \beta_{ij}(y_j - \mu_j)$, where $\beta_{ij} = \rho_{ij}\frac{\sigma_i}{\sigma_j}$ measures the degree of dependence of Y_i on Y_j in terms of their correlation coefficient ρ_{ij}. However, the conditional variance of Y_i is $\sigma_{i|j}^2 = \sigma_i^2(1 - \rho_{ij}^2)$ and is constant.

It is well known that marginal inference – computing the marginal probability distribution over a random variable given evidence – in CLGs can be performed in linear time in the size of the network, using a variable elimination algorithm in which all continuous variables are eliminated before the discrete variables (using an ordering from leaves to the parents). Thus, a CLG is a tractable model. Fig. 6.1 shows a toy tree CLG over three variables.

Next, we describe how tree CLGs can be used to construct a tractable model, by interleaving sum nodes with product nodes in a rooted directed acyclic graph, with tree CLGs as leaves.

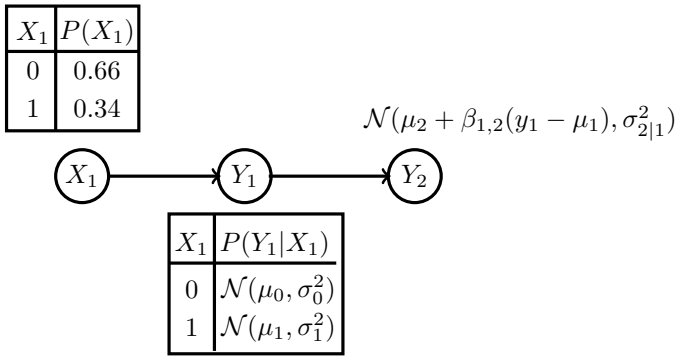


Figure 6.1. A tree CLG over discrete variable X_1 ranging over domain $\{0, 1\}$ and continuous variables Y_1 and Y_2.

6.3 Representation

In this section, we describe our new tractable model called hybrid sum-product-cutset network (HSPCN). We will begin by describing a special class of HSPCNs called hybrid cutset networks (HCNs), which do not have product or decomposition nodes.

6.3.1 Hybrid Cutset Networks

A hybrid cutset network (HCN) is a cutset network in which the OR search tree is defined over a subset of discrete variables and the tree Bayesian networks are tree CLGs. The probability distribution represented by a HCN defined over discrete variables $\mathbf{X}$ and continuous variables $\mathbf{Y}$ is given by:

$$P(\mathbf{x}, \mathbf{y}) = \mathrm{T}_{l(\mathbf{x})}(\mathbf{x}_{Scope(\mathrm{T}_{l(\mathbf{x})})}, \mathbf{y}) \prod_{(u,v) \in path(\mathbf{x})} \omega(u, v)$$

where $\omega(u, v)$ is the edge weight between nodes u and v on the unique path from the root to leaf $l(\mathbf{x})$ consistent with assignment $\mathbf{x}$ in the HCN, $\mathrm{T}_{l(\mathbf{x})}$ is the probability distribution represented by the tree CLG over variables $Scope(\mathrm{T}_{l(x)}) \subseteq \mathbf{X}$ and $\mathbf{Y}$, and $\mathbf{x}_{Scope(\mathrm{T}_{l(x)})}$ is the projection of $\mathbf{x}$ on $Scope(\mathrm{T}_{l(x)})$.[1]

[1]Although, we have defined cutset networks as having rooted OR search trees, we can easily generalize the definition to include rooted OR search graphs instead of search trees. Graphs can be obtained from OR search trees by merging identical sub-trees as described in Chapter 5.

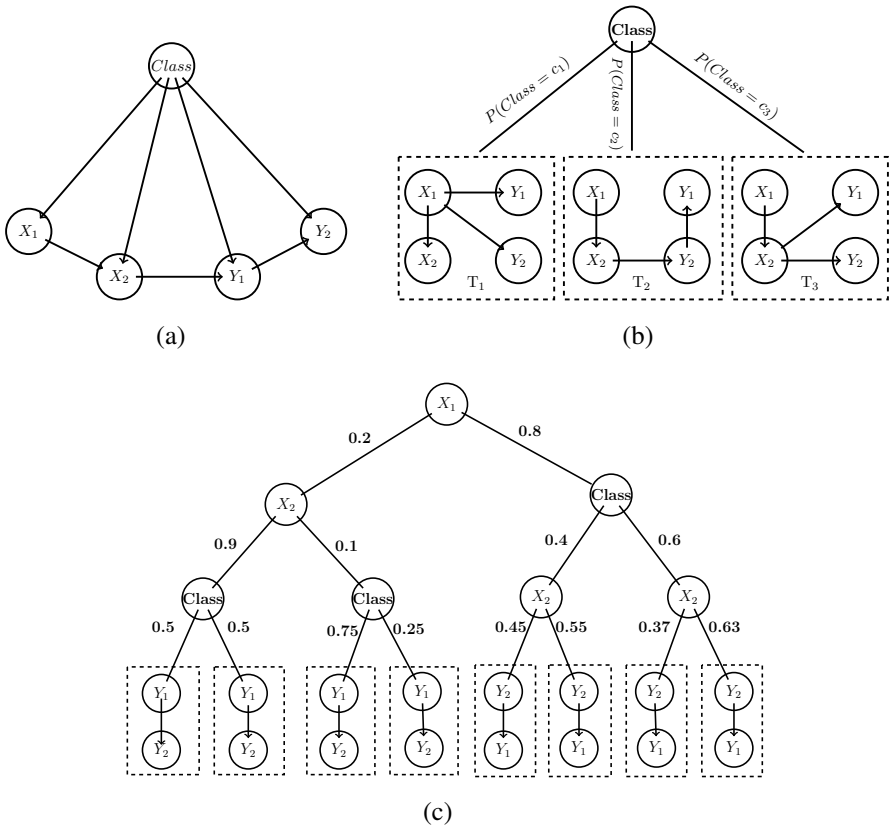


Figure 6.2. (a)Hybrid TAN and (b)Hybrid BMN and (c) Hybrid CN over discrete variables $\{X_1, X_2, Class\}$ and continuous variables $\{Y_1, Y_2\}$

.

HCNs generalize several existing classes of tractable hybrid models. For example, Tree Augmented Naive Bayes (TAN) and Bayesian Multinets (BMNs) are special cases of hybrid cutset networks in which the class variable is the only cutset variable. TAN classifiers have identical tree BN structures for each class while BMNs generalize TAN classifiers by learning a different tree BN for each class value (see Figure 6.2). Note that in a HCN the $Class$ variable is not required to be the root of the OR search tree. The key advantage of cutset networks is that they can compactly represent large treewidth graphical models. This is because one can use different tree networks at the leaves. The corresponding graphical model has an edge between two nodes if there is an edge between the nodes in any of the tree Bayesian networks, and thus can have arbitrarily large treewidth.

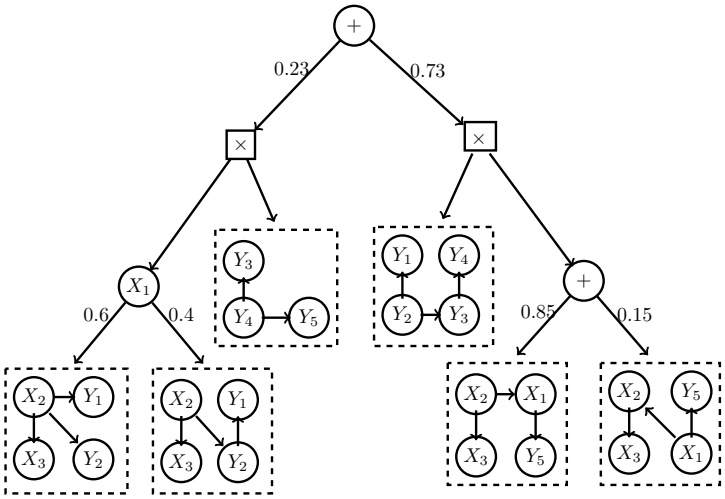


Figure 6.3. An HSPCN rooted by a latent-sum node over discrete variables $\{X_1, X_2, X_3\}$ and continuous variables $\{Y_1, Y_2, Y_3, Y_4, Y_5\}$. X_1 is the only observed sum node. Each product node decomposes the model differently and each leaf is a tree CLG over remaining observed variables.

6.3.2 Hybrid Sum-Product Cutset Networks (HSPCNs)

Hybrid sum-product-cutset networks are SPCNs introduced in Section 5.3 representing joint distributions over both discrete and continuous variables. HSPCNs are rooted directed acyclic graphs which consists of three types of nodes: latent-sum, observed-sum, and product nodes with tree CLGs at the leaves. Latent-sum nodes represent mixtures, and observed-sum nodes represent conditioning of observed discrete variables. Product nodes partition the sets $\mathbf{X}$ and $\mathbf{Y}$ into independent subsets. Figure 6.3 shows an HSPCN for a toy problem. The HSPCN is defined over two latent variables ($+$ nodes in the figure), three discrete variables and five continuous variables.

6.4 Learning HSPCNs

Algorithm 6 describes the learning algorithm for HSPCNs. The algorithm has three main steps: (1) Base case; (2) Decomposition Step and (3) Splitting Step. The algorithm recursively partitions the training examples (sum nodes) or the variables (product nodes) in the training data into subsets until a termination criteria is satisfied (base case). In the base case, we learn a tree CLG using an algorithm proposed by Friedman et al. (Friedman et al., 1998). The latter is a simple modifica-

tion of the Chow-Liu algorithm for discrete variables to CLGs. Algorithm 6 provides a general template for learning various hybrid models presented in Table 6.1. Both HNBs and HBMNs are essentially unit depth cutset networks with the dedicated class variable as the root and CLGs at the leaves. A HNB assumes complete conditional independence of the variables given the class where discrete variables are multinomials and continuous variables are simple Gaussians whereas a HBMN assumes a structurally and parametrically different tree CLG for each value of the class attribute.

Table 6.1. Structural comparison of various hybrid models. HNB:=Hybrid Naive Bayes, HBMN:= Hybrid Bayesian Multinet, HCN:= Hybrid Cutset Network, HSPN:= Hybrid Sum-Product Network, and HSPCN:= Hybrid Sum-Product-Cutset Network.

Model	**Observed Sum Node**	**Latent Sum Node**	**Product Node**	**Base Model**
HNB	Class	None	None	Univariate distributions
HBMN	Class	None	None	Tree CLG
HCN	Heuristically chosen	None	None	Tree CLG
HSPN	None	Heuristically chosen	Independence tests	Univariate distributions
HSPCN	Heuristically chosen	Heuristically chosen	Independence tests	Tree CLG

6.4.1 Learning Tree Structured CLGs

(Friedman et al., 1998) proposed an algorithm for learning tree-augmented Bayesian network classifiers over discrete and continuous random variables. Their algorithm learns structurally identical tree CLGs for each value of the class attribute by maximizing a score that depends on the conditional mutual information between variables given the class attribute. We generalize their method to learn a tree CLG based on the mutual information between variables.

Given a set of N training samples $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^N\}$ and a set of m random variables over mixed domains $\boldsymbol{V} = \boldsymbol{X} \cup \boldsymbol{Y}$, the objective is to find a tree CLG network $\mathcal{B}$ that maximizes the log-likelihood:

Algorithm 6: LEARNHSPCN($\mathcal{D}$,$\mathbf{X}$, $\mathbf{Y}$)

begin
  // 1. Base Case
  if termination conditions are satisfied **then**
    return TreeCLG($\mathcal{D}$, $\mathbf{X}$, $\mathbf{Y}$)
  end
  // 2. Decomposition Step
  if $\mathbf{X}$ and $\mathbf{Y}$ can be decomposed into subsets $\{\mathbf{X}_1, \ldots, \mathbf{X}_k\}$ and $\{\mathbf{Y}_1, \ldots, \mathbf{Y}_k\}$ respectively **then**
    return $\prod_{i=1}^{k}$ LEARNHSPCN($\mathcal{D}$,$\mathbf{X}_i$, $\mathbf{Y}_i$)
  end
  // 3. Splitting Step
  Heuristically select either a latent or an observed discrete variable from $\mathbf{X}$ to condition on.
  if *an observed variable X_i is selected* **then**
    return $\sum_j \frac{|\mathcal{D}_j|}{|\mathcal{D}|}$ LEARNHSPCN($\mathcal{D}_j$,$\mathbf{X}_{-i}$, $\mathbf{Y}$)
    where $\mathbf{X}_{-i} = \mathbf{X} \setminus \{X_i\}$ and $\mathcal{D}_j$ are the set of data points in which X_i is assigned to $x_{i,j}$
  end
  if *a latent variable is selected* **then**
    Use a clustering algorithm or hard-EM to divide the examples into k clusters.
    Let $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ be the clusters.
    return $\sum_{j=1}^{k} \frac{|\mathcal{D}_j|}{|\mathcal{D}|}$ LEARNHSPCN($\mathcal{D}_j$,$\mathbf{X}$, $\mathbf{Y}$)
  end
end

$$\begin{aligned} \mathcal{L}(\mathcal{B}; \boldsymbol{\mathcal{D}}) &= \sum_{v \in \boldsymbol{V}} \sum_{i=1}^{N} \log P_{\mathcal{B}}(x_v^i | x_{\pi(v)}^i) \\ &= \Big(\sum_{v \neq v_{root}} Score_{\mathcal{B}}(\pi(v) \rightarrow v) + Score_{\mathcal{B}}(v_{root}) \Big) \end{aligned} \tag{6.1}$$

where x_v^i is the value of variable v in sample $\boldsymbol{x}^i$, $\pi(v)$ is its parent and therefore $x_{\pi(v)}^i$ is the value of the parent $\pi(v)$ in sample $\boldsymbol{x}^i$. v_{root} is the root node of the tree $\mathcal{G}$. The problem is to essentially find the set of parents π (directed edges) of variables satisfying the constraints of a CLG such that 6.1 is maximized. Following is a graph-theoretic procedure that maximizes 6.1.

1. Initialize $\mathcal{G} = \{\boldsymbol{V}, \boldsymbol{E} = \{\}\}$

2. For each pair of variables $\langle A, B \rangle$, if $A \rightarrow B$ is valid then $\boldsymbol{E} = \boldsymbol{E} \cup \{A \rightarrow B\}$ with weight $Score(A \rightarrow B)$. The optimal CPD parameters are the maximum likelihood parameters computed from the data using standard procedures.

3. Once all $O(m^2)$ arcs are added to $\mathcal{G}$, find a set of arcs $\mathcal{S}$ that induces a maximum weighted directed spanning tree of $\mathcal{G}$ (Tarjan, 1977; Even, 1979) such that each node has at most one incoming edge.

4. For each variable B, select the CPD in step 2 that matches an arc in $\mathcal{S}$.

There are three possible cases to consider while computing $Score(A \rightarrow B)$ and in all three cases it can be shown that the score depends on the empirical mutual information $\hat{I}(A, B)$ between the variables in consideration.

1. **Both A and B are discrete:** Let $A \in \boldsymbol{X}$ defined over domain Δ_A and $B \in \boldsymbol{X}$ defined over domain Δ_B. The CPD of the B given A is modeled as a conditional probability table which defines the conditional probability of each possible value $b \in \Delta_B$ given each possible value $a \in \Delta_A$. We denote by N_{ab} as the number of samples in $\boldsymbol{\mathcal{D}}$ with $A = a$ and $B = b$. Then,

$$\begin{aligned}
Score(A \rightarrow B) &= \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} N_{ab} \log P(b|a) \\
&= N \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} \hat{P}(a, b) \log P(b|a) \\
&= N \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} \hat{P}(a, b) \log \big(\frac{P(a, b)}{P(a)P(b)} P(b) \big) \\
&= N \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} \Big(\hat{P}(a, b) \log \frac{P(a, b)}{P(a)P(b)} + \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} \hat{P}(a, b) \log P(b) \Big) \\
&= N \sum_{a \in \Delta_A} \sum_{b \in \Delta_B} \hat{P}(a, b) \log \frac{P(a, b)}{P(a)P(b)} + N \sum_{b \in \Delta_B} \hat{P}(b) \log P(b)
\end{aligned}$$

which is maximized when the marginals $P(a, b)$, $P(a)$ and $P(b)$ are equal to their empirical values. After some more algebraic manipulations $Score(A \rightarrow B)$ becomes $N\hat{I}(A, B) -$

$N\hat{H}(B)$, where $\hat{I}(A,B)$ is the empirical mutual information between A and B and $\hat{H}(B)$ is the empirical entropy of B which is independent of the data. Therefore, the best score for the edge $A \rightarrow B$ is proportional to the empirical mutual information $\hat{I}(A,B)$ between the discrete variables A and B. This idea has been used by Chow and Liu (Chow and Liu, 1968) to learn the optimal tree distribution over a set of $discrete$ random variables.

2. **Both A and B are continuous:** Let $A \in \boldsymbol{Y}$ with mean and variance μ_A and σ_A^2 respectively and let $B \in \boldsymbol{Y}$ with mean and variance μ_B and σ_B^2 respectively. The CPD of B given A is modeled as a linear Gaussian with conditional mean $\mu_{B|A} = \mu_B + \beta_{AB}(\boldsymbol{x}_A - \mu_A)$ where $\beta_{AB} = \rho_{AB}\frac{\sigma_B}{\sigma_A}$ and conditional variance $\sigma_{B|A}^2 = \sigma_B^2(1-\rho_{AB}^2)$ where ρ_{AB} is the correlation coefficient between A and B measured as $\frac{Cov(A,B)}{\sigma_A\sigma_B}$. Then,

$$Score(A \rightarrow B) = \sum_{i=1}^{N} \log P(\boldsymbol{x}_B^i|\boldsymbol{x}_A^i) = \sum_{i}^{N} \log\Big(\frac{1}{\sqrt{2\pi\sigma_{B|A}^2}} e^{(-\frac{(\boldsymbol{x}_B^i - \mu_{B|A})^2}{2\sigma_{B|A}^2})}\Big)$$

where the maximum likelihood parameters for the means (conditional and unconditional), variances (conditional and unconditional) and covariances are estimated from data using standard procedure. We denote each such parameter with a $\hat{}$ symbol. Therefore,

$$\begin{aligned}
Score(A \rightarrow B) &= N\log(\frac{1}{\sqrt{2\pi\hat{\sigma}_{B|A}^2}}) - \frac{1}{2\hat{\sigma}_{B|A}^2}\sum_{i}^{N}(\boldsymbol{x}_B^i - \hat{\mu}_{B|A})^2 \\
&= N\log(\frac{1}{\sqrt{2\pi\hat{\sigma}_{B|A}^2}}) - \frac{N\hat{\sigma}_{B|A}^2}{2\hat{\sigma}_{B|A}^2} = N\log(\frac{1}{\sqrt{2\pi e\hat{\sigma}_{B|A}^2}}) \\
&= -\frac{N}{2}\Big[\log(1-\hat{\rho}_{AB}^2) + \log(2\pi e\hat{\sigma}_B^2)\Big] \\
&= -\frac{N}{2}\log(1-\hat{\rho}_{AB}^2) - \frac{N}{2}\log(2\pi e\hat{\sigma}_B^2) = N\hat{I}(A,B) - N\hat{H}(B)
\end{aligned}$$

where $\hat{H}(B) = \frac{1}{2}\log(2\pi e\hat{\sigma}_B^2)$ is the empirical differential entropy of the continuous variable B which is independent of the data and $\hat{I}(A,B) = -\frac{1}{2}\log(1-\hat{\rho}_{AB}^2)$ is empirical mutual information between the Gaussian variables A and B.

3. A **is discrete and** B **is continuous:** Let $A \in \boldsymbol{X}$ with domain Δ_A and $B \in \boldsymbol{Y}$. For each value $a \in \Delta_A$, B is normally distributed with the conditional mean $\mu_{B|a}$ and conditional variance $\sigma^2_{B|a}$. As before, in the following we assume maximum likelihood parameters to compute the score of $A \to B$ and denote them with a ˆ symbol.

$$
\begin{aligned}
Score(A \to B) &= \sum_{a \in \Delta_A} \sum_{j=1}^{\boldsymbol{\mathcal{D}}_{A=a}} \log P(\boldsymbol{x}_B^j | \boldsymbol{x}_A^j) \\
&= \sum_{a \in \Delta_A} \sum_{j=1}^{\boldsymbol{\mathcal{D}}_{A=a}} \log \left(\frac{1}{\sqrt{2\pi\hat{\sigma}^2_{B|a}}} e^{\left(-\frac{(\boldsymbol{x}_B^j - \hat{\mu}_{B|a})^2}{2\hat{\sigma}^2_{B|a}} \right)} \right) \\
&= \sum_{a \in \Delta_A} \left[N_a \log\Big(\frac{1}{\sqrt{2\pi\hat{\sigma}^2_{B|a}}} \Big) - \frac{1}{2\hat{\sigma}^2_{B|a}} \sum_{j}^{\boldsymbol{\mathcal{D}}_{A=a}} (\boldsymbol{x}_B^j - \hat{\mu}_{B|a})^2 \right] \\
&= \sum_{a \in \Delta_A} \left[N_a \log\Big(\frac{1}{\sqrt{2\pi\hat{\sigma}^2_{B|a}}} \Big) - \frac{N_a}{2} \right] \\
&= \sum_{a \in \Delta_A} -\frac{N_a}{2} \log(2\pi e \hat{\sigma}^2_{B|a}) \\
&= -\frac{N}{2} \sum_{a \in \Delta_A} \hat{P}(a) \log(2\pi e \hat{\sigma}^2_{B|a}) \\
&= -\frac{N}{2} \left[\sum_{a \in \Delta_A} \hat{P}(a) \log(2\pi e) + \sum_{a \in \Delta_A} \hat{P}(a) \log(\hat{\sigma}^2_{B|a}) \right] \\
&= -\frac{N}{2} \Big[\log(2\pi e \hat{\sigma}^2_B) - \log(\hat{\sigma}^2_B) + \sum_{a \in \Delta_A} \hat{P}(a) \log(\hat{\sigma}^2_{B|a}) \Big] \\
&= \frac{N}{2} \Big[\log(\hat{\sigma}^2_B) - \sum_{a \in \Delta_A} \hat{P}(a) \log(\hat{\sigma}^2_{B|a}) \Big] - N[\frac{1}{2} \log(2\pi e \hat{\sigma}^2_B)] \\
&= N\hat{I}(A, B) - N\hat{H}(B)
\end{aligned}
$$

where $\hat{\sigma}^2_B$ is the empirical variance of B and $\hat{I}(A, B) = \frac{1}{2} \Big[\log(\hat{\sigma}^2_B) - \sum_{a \in \Delta_A} \hat{P}(a) \log(\hat{\sigma}^2_{B|a}) \Big]$ is the empirical mutual information between discrete variable A and the Gaussian variable B (Martínez et al., 2006).

The above scores can be computed by a single pass over the data. Hence, learning tree CLGs can be done efficiently.

Next, we describe the decomposition and splitting steps in turn.

6.4.2 Decomposition

We use an approach similar to Gens and Domingos (Gens and Domingos, 2013) to decompose variables into independent components. The key idea is to compute a mutual information graph over the discrete and continuous variables. The graph has variables as nodes and a weighted edge between all pairs of variables (it is a complete graph). Mutual information measures the degree of statistical dependence between two variables. Mutual information between pairs of discrete and pairs of continuous variables is defined in the usual way. We use the technique given in Proposition 1 of (Martínez et al., 2006) to compute mutual information between a discrete and a continuous variable. We then prune any *weak edges* from the mutual information graph; edges whose weight falls below a predefined threshold. The connected components of the resultant graph produce the desired decomposition.

6.4.3 Splitting

In the splitting step, we heuristically select either a latent variable or an observed variable to condition on. To select a latent variable, we use the following method. We partition the training instances using EM with a $K-$component mixture model rooted by a latent sum node having a tree CLG in each of its K components. Once EM converges, we obtain a weighted (soft EM) or unweighted (hard EM) partition of the data. Each edge from the latent root sum node is labeled by the fraction of instances belonging to that component of the mixture. The number of K components in the mixture can be chosen by an MDL/BIC scoring function as in (Friedman et al., 1997) with a penalty $3K\frac{\log(N)}{2}$, where N is the number of training instances.

To select an observed variable, we use the following heuristic based on maximum likelihood. For each discrete variable $X_i \in \mathbf{X}$, we learn a mixture model rooted by X_i such that for each of its values $x_{i,j}$ in its domain, the mixture has a tree CLG over $\{\mathbf{X} \setminus X_i\} \cup \mathbf{Y}$ learned using data $\boldsymbol{\mathcal{D}}_j$ which is the current data $\boldsymbol{\mathcal{D}}$ conditioned on $X_i = x_{i,j}$. Each edge from the root sum node is labeled by the fraction of training instances belonging to that split. Each X_i is scored according to

the likelihood of the mixture model that it roots and the highest scoring X_i is then chosen as the desired variable to split the data.

To decide between an $observed$ sum node (cutset conditioning) and a $latent$ sum node (mixture) in the splitting step, we use the MDL/BIC scoring method with a penalty of $|\mathcal{M}|\frac{\log(N)}{2}$, where $|\mathcal{M}|$ is the size (number of free parameters) of the model $\mathcal{M}$. If the score on the observed variable is higher than the latent variable, we perform splitting using an observed variable. Otherwise, we split on a latent variable.

6.5 Experiments

We evaluated the performance of HSPCNs as density estimators as well as probabilistic classifiers over mixed domains. Table 6.2 shows the 18 real world datasets from various domains with mixed variables that were chosen from the UCI machine learning repository (Merz et al., 1997). All the problems have a designated class attribute which is considered as a discrete valued variable in this dissertation. We used the training and test sets for each problem available from the MLC++ (Kohavi et al., 1994) website. Samples with missing values were discarded. We dedicated the first 10% samples from the training set as validation set to choose the tuning parameters for the models. The best parameters were chosen based on maximum validation set log-likelihood and used for learning the final model using the complete training set. All the reported results are based on the final model's performance on the test set.

For a diverse comparison we trained five different models over the hybrid domains namely: (1) Hybrid Naive Bayes (HNB), (2) Hybrid Bayesian Multinets (HBMNs), (3) Hybrid Cutset Networks (HCNs), (4) Hybrid Sum-Product Networks (HSPNs) and (5) Hybrid Sum-Product-Cutset Networks (HSPCNs). For HCNs, HSPNs and HSPCNs, the depth was varied from 1 to 10 and the best depth was chosen based on the log-likelihood of the validation set. A tree CLG was learned whenever the max depth was reached or the number of training samples fell below 5. For learning latent sum nodes in HSPNs and HSPCNs, we ran hard EM for 10 iterations or until convergence

to cluster instances. To avoid multiple random restarts in EM, we ran WEKA's (Holmes et al., 1994) implementation of KMeans clustering to cluster samples and then use the samples in each cluster to learn the initial parameters of each mixture component. The number of mixture components were varied from 2 to 8 with increments of 2 and the best number of components were chosen based on the BIC/MDL score with the penalty mentioned in Section 6.4.3. Decompositions were approximated by pruning low mutual information edges from the mutual information graph. Three different sets of thresholds were used: (1) $\{0.0001, 0.01, 0.1\}$ to prune edges between two discrete variables, (2) $\{0.0002, 0.02, 0.1\}$ to prune edges between two continuous variables and (3) $\{0.0003, 0.03, 0.1\}$ to prune edges between discrete and continuous variable. The smoothing technique proposed in (Friedman et al., 1998) with value 1.0 was applied to compute smoothed estimates of means and variances of continuous variables and smoothed parameters of CPT values of all discrete variables.

6.5.1 Density Estimation

Table 6.2 presents the average log-likelihood scores of various models evaluated on the test set. HNBs is the weakest model with the lowest average score of -24.77 per dataset. HCNs perform better than HNBs and HBMNs in most of the cases except for the datasets `"australian", "cleve" and "german"` in which the models overfit with increased number of cutsets. HSPNs have levels of latent sum and product nodes with complete independence over the observed variables at the leaves (Gens and Domingos, 2013). HSPNs perform better than HNBs, HBMNs and HCNs in 17 out of 18 cases. HSPNs perform better than HSPCNs in 8 out of 18 cases. HSPCNs perform better than any other models by scoring the highest -19.50 on average. HSPCNs always score significantly better than HNBs, HBMNs and HCNs and sometimes slightly worse than HSPNs in cases where the underlying model is simpler and/or sufficient data is lacking to learn accurate observed sum nodes. This can be observed for the datasets `"australian", "cleve", "echocardiogram"` and `"german"`.

Table 6.2. Comparison of average test set log-likelihood scores. Columns $|Train|$, $|Valid|$ and $|Test|$ provides the number of samples for training, validation and test set respectively. Columns labeled D and C indicate the number of discrete and continuous valued variables present in the domain respectively. Bold values indicate the highest average score achieved by a model.

Datasets	$\|Train\|$	$\|Valid\|$	$\|Test\|$	D	C	HNB	HBMN	HCN	HSPN	HSPCN
anneal-U	539	59	300	33	6	-40.87	-32.84	-32.22	-31.54	-31.08
australian	414	46	230	9	6	-36.26	-36.66	-37.85	**-31.75**	-32.00
auto	97	10	52	11	15	-70.40	-66.47	-64.53	-61.55	**-60.06**
balance-scale	375	41	209	1	4	-7.48	-7.47	-7.47	**-4.98**	**-4.98**
breast	412	45	226	1	10	-31.25	-30.70	-30.70	-25.07	**-24.46**
cars1	235	26	131	1	7	-28.96	-27.38	-27.38	-25.14	**-25.09**
cleve	178	19	99	8	6	-25.81	-25.97	-26.80	**-24.97**	-25.08
crx	419	46	188	10	6	-34.87	-33.04	-33.04	**-31.86**	-31.87
diabetes	461	51	256	1	8	-30.34	-29.97	-29.97	**-27.57**	-27.59
echocardiogram	66	7	34	2	6	-11.44	-11.61	-11.61	**-11.07**	-11.44
flare	639	71	356	9	2	-7.25	-6.35	-5.25	-2.69	**-2.36**
german	600	66	334	14	7	-34.79	-33.50	-35.32	**-25.40**	-26.21
german-org	600	66	334	13	12	-30.39	-29.15	-28.94	-15.73	**-15.70**
glass	128	14	72	1	9	-2.79	-1.70	-1.70	1.37	**1.77**
glass2	98	10	55	1	9	1.63	2.49	2.49	6.49	**7.13**
heart	162	18	90	1	13	-28.65	-28.31	-28.31	**-17.90**	-18.08
iris	90	10	50	1	4	-2.59	**-2.24**	**-2.24**	-2.52	**-2.24**
liver-disorder	207	23	115	1	6	-23.40	-23.00	-23.00	**-21.47**	-21.64
Average LL						-24.77	-23.55	-23.55	-19.63	**-19.50**

6.5.2 Classification

Table 6.3 presents the performance of the models as classifiers. HSPCNs have superior scores compared to other models in 8 out of the 18 datasets. They have on average an accuracy of more than 80% which is significantly higher than other models. In general, the performance of HSPCNs lies between the performances of HCNs and HSPNs and always closer to the better performing one. The weakest classifier in terms of average accuracy obtained per dataset is the hybrid naive Bayes and it scores better than others only in 2 out of the 18 cases. As in density estimation, HCNs rarely perform worse than HBMNs, exceptions being the `"german"` and `"german-org"` datasets where simpler models dominate the competition. Unlike density estimation, HSPNs perform better than other only in 1 out of the 18 datasets.

Table 6.3. Comparison of classification accuracy evaluated on the test set. Bold values indicate the best accuracy achieved by the model on the particular instance.

Datasets	HNB	HBMN	HCN	HSPN	HSPCN
anneal-U	81.67	93.33	94.67	83.33	**96.33**
australian	77.83	77.83	82.17	82.17	**83.48**
auto	46.15	65.38	**69.23**	50.00	63.46
balance-scale	**89.47**	88.52	88.52	56.94	89.47
breast	96.46	95.58	95.58	97.35	**97.79**
cars1	62.60	62.60	62.60	68.70	**69.47**
cleve	81.82	75.76	75.76	**82.83**	81.82
crx	78.19	78.19	78.19	82.45	**83.51**
diabetes	73.44	**74.22**	**74.22**	69.14	73.44
echocardiogram	**79.41**	**79.41**	**79.41**	76.47	79.41
flare	81.18	75.56	82.87	**85.11**	85.11
german	**76.95**	73.05	69.76	72.75	72.75
german-org	74.85	**77.25**	72.16	72.75	72.75
glass	50.00	50.00	50.00	**69.44**	69.44
glass2	61.82	65.45	65.45	81.82	**87.27**
heart	85.56	**86.67**	**86.67**	83.33	85.56
iris	94.00	96.00	96.00	96.00	**98.00**
liver-disorder	55.65	57.39	57.39	61.74	**63.48**
Average Accuracy	74.84	76.23	76.70	76.24	**80.70**

6.6 Chapter Summary

In this chapter, we proposed hybrid sum-product-cutset networks, which can represent high-dimensional joint distributions over both discrete and continuous random variables. Our proposed algorithm for learning the structure of HSPCNs can be considered as the first advanced structure learning algorithm for SPNs and CNs with continuous variables present in the domain. A thorough evaluation of the performance of HSPCNs on several real-world problems with a good mix of discrete and continuous variables revealed that HSPCNs are superior in general in modeling the underlying distributions than standalone CNs, SPNs and other state-of-the-art tractable PGMs.

CHAPTER 7

CONCLUSION

This final chapter concludes this dissertation by highlighting our contributions and providing valuable insights for potential future work.

7.1 Contributions

The goal of the research presented in this dissertation has been two-fold: proposing novel and tractable representations for modeling the joint distributions over random variables and devising efficient algorithms for learning these models form data.

7.1.1 Proposed Tractable PGMs

We have proposed the following three different tractable models.

- **Cutset Networks (CNs)** – our first proposed models which combine an OR search tree with tree Bayesian or Markov networks. CNs are essentially a hybrid model which combine an inference representation with a tractable (tree) PGM. CNs have two attractive properties: they are highly interpretable and they often yield a compact representation by exploiting various symmetry properties such as context-specific independence, identical probability values and determinism (Gogate and Domingos, 2010b; Gogate, 2009).

- **Sum-Product-Cutset Networks (SPCNs)** – combines sum-product networks (SPNs) with CNs. Our experimental evaluations revealed that flat mixtures of CNs without any model decompositions significantly improved the prediction accuracy. SPNs achieve high accuracy by using both mixture (sum) and decomposition (product) nodes; the product nodes partition

the variables into independent components. Therefore, we hypothesized that by combining SPNs with CNs via a new deep architecture called SPCNs, we can further improve the accuracy of flat mixtures of CNs, by exploiting problem decomposition.

- **Hybrid Sum-Product-Cutset Networks (HSPCNs)** – a simple and natural extension of SPCNs to model distributions over domains having both discrete and continuous random variables. The leaf distributions in HSPCN are conditional linear Gaussians to maintain tractable inference and efficient learning.

7.1.2 Proposed Learning Algorithms

For each novel representation, we proposed several advanced algorithms for learning both its structure and the parameters from data.

- **Learning Cutset Networks**

  We proposed an efficient algorithm for learning CNs from data. The algorithm recursively partitions the data based on some heuristic choice and then uses the celebrated Chow-Liu algorithm to induce a tree Bayesian or Markov network. Our proposed algorithm is simple, easily implementable and general enough to be used with any suitable heuristic.

- **Learning Ensembles of Cutset Networks**

  We proposed several sequential boosting based and parallel bagging based methods for learning ensembles of CNs. Our experimental evaluations showed that, unlike learning mixtures of CNs in a slow iterative manner, our proposed ensemble techniques can learn much more accurate mixtures and much faster.

- **Learning Sum-Product-Cutset Networks**

  Both SPN and CN learning algorithms are recursive in nature and share some common steps such as inducing a base model and splitting training instances to learn sum nodes. We proposed an algorithm for learning SPCNs that follows similar steps as an SPN and CN learner,

but also allows a mechanism to choose between an observed sum node (cutset conditioning) or a latent sum node (mixture) at each recursive step. The final result is that the algorithm provides a general template for learning an array of models – CNs, SPNs, and SPCNs.

- **Learning Hybrid Sum-Product-Cutset Networks**

  Our proposed hybrid model learning algorithm learns a conditional linear Gaussian distribution at the leaves of CNs and SPCNs and also finds approximate decompositions over mixed variables. As mentioned before, this is the first advanced algorithm for learning the structure and parameters of a deep architecture such as SPN over mixed domains.

- **Learning to Merge**

  To date, all algorithms proposed in literature for learning SPNs induce models which are directed trees – there is a single path from the root to each node in the network. Our final contribution in learning models is that we devised efficient post-optimization strategies for finding similar sub-networks (sub-CNs, sub-SPNs, sub-SPCNs and their hybrids) in the tree structured model and then merge them to form a more compact and accurate graph structured model.

We evaluated the performance of each of our proposed models and learning algorithms on a wide variety of high-dimensional real-world datasets, chosen from different domains. Our results clearly show that the predictive accuracy, learning and prediction times of our new models and learning algorithms are superior than state-of-the-art tractable model learners in literature.

7.2 Future Work

7.2.1 Discriminative Learning of CNs

The learning algorithms presented in this dissertation learn tractable models in a generative fashion by optimizing log-likelihood. The learned models may not accurately capture the conditional distribution of query variables given the evidence variables in domains where the evidence is known

a priori. Discriminative learning has been extensively studied in the context of Bayesian network classifiers. (Greiner and Zhou, 2002) proposed a discriminative parameter learning maximizing the conditional likelihood of Bayesian network classifiers given the structure. (Grossman and Domingos, 2004) proposed a discriminative structure learning technique for Bayesian network classifiers maximizing the conditional likelihood. However the parameters were learned by maximum likelihood estimation. (Keogh and Pazzani, 1999; Pernkopf, 2005) used classification rate as the objective function to discriminatively learn the structure and parameters of BN classifiers and (Bilmes and Morgan, 1999; Bilmes, 2000; Pernkopf and Bilmes, 2005) proposed explaining away residual for discriminative structure learning of dynamic BNs. Discriminative structure learning has recently been studied by (Rooshenas and Lowd, 2016) only for arithmetic circuits over discrete domains and discriminative parameter learning by (Gens and Domingos, 2012) for SPNs. An important future research direction is thus to discriminatively learn both the structure and parameters of CNs, SPCNs and HSPCNs. Such learning will help to build models that have more accurate conditional distributions and as a result allow them to be compared against other well-known discriminative learners like SVMs, logistic regression and neural networks for supervised classification tasks.

7.2.2 Dynamic Discritization of Continuous Variables in Hybrid Models

In this dissertation, we proposed hybrid models which allow conditioning over discrete variables only. A common approach to deal with mixed domains is to discretize all the continuous attributes as a pre-processing step of learning. This often significantly reduces the accuracy of models due to loss of information. (Friedman et al., 1998) has proposed hybrid tree-augmented naive Bayes models in which dual representation is possible where continuous variables can retain both their original and discretized versions. Instead of discretizing every continuous variable in advance or maintaining a dual representation of them, in CNs and SPCNs one has the provision to dynamically choose a continuous attribute that is best represented discretized, and thoroughly investigating this is an interesting line of future research.

7.2.3 Learning More Expressive Base Models

CNs and SPCNs were designed to allow only tree width one models at the leaves. These models are often too biased in the presence of large amounts of data. One possible solution could be to learn deeper models. (Vergari et al., 2015) proposed to learn SPNs with tree Bayesian networks at the leaves where the depth of the SPNs was widely varied from 15 to 59. However, as we increase the depth, it causes an exponential decrease in the number of samples available at the leaves, which in turn causes the model to overfit the data because of high variance. This can be remedied by learning ensembles of SPNs as done by (Vergari et al., 2015). But such an approach causes a significant increase in the learning time. An alternative solution is to learn complex models. (Rooshenas and Lowd, 2014) proposed to learn SPNs with arithmetic circuits at the leaves when fewer than 50 samples were available. These models out-perform the models proposed by (Vergari et al., 2015) in cases where sufficient data was available while they performed poorly when data was scarce. A more effective solution worth future investigation to avoid high bias associated with simpler models and high variance with deeper models, is to learn expressive enough base models at the right point in learning. For example, learning polytrees with bounded number of parents or learning bounded tree width Bayesian networks.

7.2.4 Relational Merging

Merging sub-networks improves model accuracy and reduces inference time as demonstrated in this dissertation. A restriction of our proposed merging approach is that it only searches for similar sub-networks over the same scope. More often than not, real-world scenarios have similarities which span over different scopes (e.g. pixels of the left side of a facial image have the same values as the pixels in the rights side). Following are two potential future works in this context: (1) develop relational merging approaches that search for similarities in sub-networks having different (even disjoint) scopes (cf. (Gogate and Domingos, 2010a, 2011)) and (2) analyzing contexts –

assignment to variables on the path from the root – of merged sub-networks for finding symmetric contexts.

REFERENCES

Adel, T., D. Balduzzi, and A. Ghodsi (2015). Learning the structure of sum-product networks via an svd-based algorithm. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pp. 32–41.

Ammar, S., P. Leray, F. Schnitzler, et al. (2010). Subquadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models*, pp. 17–24.

Bach, F. R. and M. I. Jordan (2001). Thin junction trees. *Advances in Neural Information Processing Systems 14*, 569–576.

Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, 281–305.

Bilmes, J. and N. Morgan (1999). *Natural statistical models for automatic speech recognition*. University of California, Berkeley.

Bilmes, J. A. (2000). Dynamic bayesian multinets. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)*, pp. 38–45.

Bøttcher, S. G. (2004). *Learning Bayesian networks with mixed variables*. Ph. D. thesis, Citeseer.

Boutilier, C., N. Friedman, M. Goldszmidt, and D. Koller (1996). Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 115–123. Morgan Kaufmann.

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

Chavira, M. and A. Darwiche (2007). Compiling Bayesian networks using variable elimination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2443–2449.

Chavira, M. and A. Darwiche (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence 172*(6-7), 772–799.

Chechetka, A. and C. Guestrin (2008). Efficient principled learning of thin junction trees. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20*. MIT Press.

Choi, M. J., V. Tan, A. Anandkumar, and A. Willsky (2011, May). Learning latent tree graphical models. *Journal of Machine Learning Research 12*, 1771–1812.

Chow, C. K. and C. N. Liu (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory 14*, 462–467.

Cooper, G. F. (1990, March). The computational complexity of probabilistic inference using Bayesian belief netw orks. *Artificial Intelligence 42*(2-3), 393–405.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Darwiche, A. (2001). Decomposable negation normal form. *Journal of the ACM 48*(4), 608–647.

Darwiche, A. (2003). A differential approach to inference in Bayesian networks. *Journal of the ACM 50*(3), 280–305.

Darwiche, A. (2009). *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

Davis, J. and P. Domingos (2010). Bottom-up learning of Markov network structure. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, Haifa, Israel, pp. 271–278. ACM Press.

Dechter, R. and R. Mateescu (2007). AND/OR search spaces for graphical models. *Artificial Intelligence 171*, 73–106.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B 39*, 1–38.

Dennis, A. and D. Ventura (2012). Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pp. 2033–2041.

Dennis, A. and D. Ventura (2015). Greedy structure search for sum-product networks. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 932–938. AAAI Press.

Di Mauro, N., A. Vergari, and F. Esposito (2015). Learning accurate cutset networks by exploiting decomposability. In *AI*IA 2015 Advances in Artificial Intelligence*, pp. 221–232.

Even, S. (1979). *Graph Algorithms*. New York, NY, USA: W. H. Freeman & Co.

Freund, Y. and R. E. Schapire (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 148–156.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55*(1), 119–139.

Friedman, N., D. Geiger, and M. Goldszmidt (1997). Bayesian network classifiers. *Machine Learning 29*(2-3), 131–163.

Friedman, N. and M. Goldszmidt (1996). Discretizing continuous attributes while learning bayesian networks. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML)*, pp. 157–165.

Friedman, N., M. Goldszmidt, and T. J. Lee (1998). Bayesian network classification with continuous attributes: Getting the best of both discretization and parametric fitting. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML)*, pp. 179–187.

Geiger, D. and D. Heckerman (1996a). Beyond bayesian networks: Similarity networks and bayesian multinets. *Artificial Intelligence 82*, 45–74.

Geiger, D. and D. Heckerman (1996b). Knowledge representation and inference in similarity networks and bayesian multinets. *Artificial Intelligence 82*(1), 45–74.

Gens, R. and P. Domingos (2012). Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 3248–3256.

Gens, R. and P. Domingos (2013). Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 873–880.

Gogate, V. and P. Domingos (2010a). Exploiting Logical Structure in Lifted Probabilistic Inference. In *AAAI 2010 Workshop on Statistical Relational Learning*.

Gogate, V. and P. Domingos (2010b). Formula-Based Probabilistic Inference. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 210–219.

Gogate, V. and P. Domingos (2011). Probabilistic Theorem Proving. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 256–265. AUAI Press.

Gogate, V., W. Webb, and P. Domingos (2010). Learning efficient Markov networks. In *Proceedings of the 24th conference on Neural Information Processing Systems*, pp. 748–756.

Gogate, V. G. (2009). *Sampling algorithms for probabilistic graphical models with determinism*. Ph. D. thesis, University of California, Irvine.

Greiner, R. and W. Zhou (2002). Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 167–173. AAAI Press.

Grossman, D. and P. Domingos (2004). Learning Bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, New York, NY, USA, pp. 361–368. ACM.

Heckerman, D. and D. Geiger (1995). Learning bayesian networks: A unification for discrete and gaussian domains. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 274–284.

Holmes, G., A. Donkin, and I. H. Witten (1994). Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pp. 357–361. IEEE.

Keogh, E. J. and M. J. Pazzani (1999). Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics, AISTATS*.

Kisa, D., G. Van den Broeck, A. Choi, and A. Darwiche (2014). Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*.

Kohavi, R., G. John, R. Long, D. Manley, and K. Pfleger (1994). Mlc++: A machine learning library in c++. In *Tools with Artificial Intelligence, 1994. Proceedings., Sixth International Conference on*, pp. 740–743. IEEE.

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press.

Kozlov, A. V. and D. Koller (1997). Nonuniform dynamic discretization in hybrid networks. In *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*, pp. 314–325.

Kullback, S. and R. A. Leibler (1951). On information and sufficiency. *The annals of mathematical statistics 22*(1), 79–86.

Lauritzen, S. L. (1996). *Graphical models*, Volume 17. Clarendon Press.

Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological) 50*(2), 157–224.

Lauritzen, S. L. and N. Wermuth (1984). *Mixed interaction models*. Institut for Elektroniske Systemer, Aalborg Universitetscenter.

Lauritzen, S. L. and N. Wermuth (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics 17*(1), 31–57.

Liu, Q. and A. T. Ihler (2013). Variational algorithms for marginal MAP. *Journal of Machine Learning Research 14*(1), 3165–3200.

Lowd, D. and J. Davis (2010). Learning Markov network structure with decision trees. In *Proceedings of the 10th International Conference on Data Mining*, pp. 334–343.

Lowd, D. and P. Domingos (2008). Learning arithmetic circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, Helsinki, Finland. AUAI Press.

Lowd, D. and A. Rooshenas (2013). Learning markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pp. 406–414.

Martínez, A. P., P. Larrañaga, and I. Inza (2006). Supervised classification with conditional gaussian networks: Increasing the structure complexity from naive bayes. *Int. J. Approx. Reasoning 43*(1), 1–25.

Mateescu, R., R. Dechter, and R. Marinescu (2008). AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research 33*, 465–519.

Meila, M. and M. Jordan (2000). Learning with mixtures of trees. *Journal of Machine Learning Research 1*, 1–48.

Merz, C., P. Murphy, and D. Aha (1997). Uci repository of machine learning databases. dept. of information and computer science, univ. of california, irvine.

Mitchell, T. M. (1997). *Machine Learning*. New York, NY: McGraw-Hill.

Murphy, K. P. (1998). *Inference and learning in hybrid bayesian networks*. University of California, Berkeley, Computer Science Division.

Narasimhan, M. and J. Bilmes (2004). Pac-learning bounded tree-width graphical models. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 410–417. AUAI Press.

Nath, A. and P. M. Domingos (2015). Learning relational sum-product networks. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 2878–2886.

Neal, R. M. and G. E. Hinton (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.

Park, J. D. (2002). Map complexity results and approximation methods. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, USA, pp. 388–396. Morgan Kaufmann Publishers Inc.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Peharz, R., B. C. Geiger, and F. Pernkopf (2013). Greedy part-wise learning of sum-product networks. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II*, pp. 612–627.

Pernkopf, F. (2005). Bayesian network classifiers versus selective k-nn classifier. *Pattern Recognition 38*(1), 1–10.

Pernkopf, F. and J. Bilmes (2005). Discriminative versus generative parameter and structure learning of bayesian network classifiers. In *Proceedings of the 22nd international conference on Machine learning*, pp. 657–664. ACM.

Poon, H. and P. Domingos (2011). Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, Barcelona, Spain, pp. 337–346. AUAI Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Rahman, T. and V. Gogate (2016a). Learning ensembles of cutset networks. In *AAAI conference on Artificial Intelligence*, pp. 3301–3307.

Rahman, T. and V. Gogate (2016b). Merging strategies for sum-product networks: From trees to graphs. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016*.

Rahman, T., P. Kothalkar, and V. Gogate (2014). Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceedings of ECML and PKDD*, pp. 630–645.

Ridgeway, G. (2002). Looking for lumps: Boosting and bagging for density estimation. *Computational Statistics & Data Analysis 38*(4), 379–392.

Rooshenas, A. and D. Lowd (2013). Learning tractable graphical models using mixture of arithmetic circuits. In *Proceedings of the 17th AAAI Conference on Late-Breaking Developments in the Field of Artificial Intelligence*, pp. 104–106. AAAI Press.

Rooshenas, A. and D. Lowd (2014). Learning sum-product networks with direct and indirect interactions. In *Proceedings of the Thirty-First International Conference on Machine Learning*, Beijing, China. JMLR: W&CP 32.

Rooshenas, A. and D. Lowd (2016). Discriminative structure learning of arithmetic circuits. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1506–1514.

Rosset, S. and E. Segal (2002). Boosting density estimation. In *Advances in Neural Information Processing Systems*, pp. 641–648.

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence 82*(1), 273–302.

Silva, R., C. Blundell, and Y. W. Teh (2011). Mixed cumulative distribution networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, pp. 670–678.

Srebro, N. (2003). Maximum likelihood bounded tree-width Markov networks. *Artificial Intelligence 143*(1), 123–138.

Tang, Y., R. Salakhutdinov, and G. E. Hinton (2012). Deep mixtures of factor analysers. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.

Tarjan, R. E. (1977). Finding optimum branchings. *Networks 7*(1), 25–35.

Uria, B., I. Murray, and H. Larochelle (2013). RNADE: the real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pp. 2175–2183.

Van Haaren, J. and J. Davis (2012). Markov network structure learning: A randomized feature generation approach. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pp. 1148–1154.

Vergari, A., N. Di Mauro, and F. Esposito (2015). Simplifying, regularizing and strengthening sum-product network structure learning. In *Machine Learning and Knowledge Discovery in Databases*, pp. 343–358. Springer.

Welling, M., R. S. Zemel, and G. E. Hinton (2002). Self supervised boosting. In *Advances in Neural Information Processing Systems*, pp. 665–672.

Wiegerinck, W. (2000). Variational approximations between mean field theory and the junction tree algorithm. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 626–633.

Yang, Y. and G. I. Webb (2009). Discretization for naive-bayes learning: managing discretization bias and variance. *Machine Learning 74*(1), 39–74.

Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms* (1st ed.). Chapman & Hall/CRC.

VITA

Tahrima Rahman earned her Bachelor's degree in 2007 in Computer Science and Engineering from the University of Dhaka, Bangladesh. In 2009, she completed her Master's degree from the University of Dhaka, Bangladesh in the same discipline. Before starting her PhD in Fall 2011 at The University of Texas at Dallas, Tahrima served as a lecturer in the Institute of Information Technology, University of Dhaka. Before that she was a lecturer in the department of Computer Science and Engineering at Eastern University, Dhaka, Bangladesh for three years.