Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

Learning with Mixtures of Trees

Marina Meilă MMP@STAT.WASHINGTON.EDU
Department of Statistics
University of Washington
Seattle, WA 98195-4322, USA

Michael I. Jordan JORDAN@CS.BERKELEY.EDU
Division of Computer Science and Department of Statistics
University of California
Berkeley, CA 94720-1776, USA

Editor: Leslie Pack Kaelbling

Abstract

This paper describes the mixtures-of-trees model, a probabilistic model for discrete multidimensional domains. Mixtures-of-trees generalize the probabilistic trees of Chow and Liu (1968) in a different and complementary direction to that of Bayesian networks. We present efficient algorithms for learning mixtures-of-trees models in maximum likelihood and Bayesian frameworks. We also discuss additional efficiencies that can be obtained when data are "sparse," and we present data structures and algorithms that exploit such sparseness. Experimental results demonstrate the performance of the model for both density estimation and classification. We also discuss the sense in which tree-based classifiers perform an implicit form of feature selection, and demonstrate a resulting insensitivity to irrelevant attributes.

1. Introduction

Probabilistic inference has become a core technology in AI, largely due to developments in graph-theoretic methods for the representation and manipulation of complex probability distributions (Pearl, 1988). Whether in their guise as directed graphs (Bayesian networks) or as undirected graphs (Markov random fields), *probabilistic graphical models* have a number of virtues as representations of uncertainty and as inference engines. Graphical models allow a separation between qualitative, structural aspects of uncertain knowledge and the quantitative, parametric aspects of uncertainty—the former represented via patterns of edges in the graph and the latter represented as numerical values associated with subsets of nodes in the graph. This separation is often found to be natural by domain experts, taming some of the problems associated with structuring, interpreting, and troubleshooting the model. Even more importantly, the graph-theoretic framework has allowed for the development of general inference algorithms, which in many cases provide orders of magnitude speedups over brute-force methods (Cowell, Dawid, Lauritzen, & Spiegelhalter, 1999; Shafer & Shenoy, 1990).

These virtues have not gone unnoticed by researchers interested in machine learning, and graphical models are being widely explored as the underlying architectures in systems for classification, prediction and density estimation (Bishop, 1999; Friedman, Geiger, & Goldszmidt, 1997; Heckerman, Geiger, & Chickering, 1995; Hinton, Dayan, Frey, & Neal, 1995; Friedman, Getoor, Koller, & Pfeffer, 1996; Monti & Cooper, 1998; Saul & Jordan, 1999). Indeed, it is possible to view a wide variety of classical machine learning architectures as instances of graphical models, and the graphical model framework provides a natural design procedure for exploring architectural variations on classical themes (Buntine, 1996; Smyth, Heckerman, & Jordan, 1997).

As in many machine learning problems, the problem of learning a graphical model from data can be divided into the problem of *parameter learning* and the problem of *structure learning*. Much progress has been made on the former problem, much of it cast within the framework of the expectation-maximization (EM) algorithm (Lauritzen, 1995). The EM algorithm essentially runs a probabilistic inference algorithm as a subroutine to compute the "expected sufficient statistics" for the data, reducing the parameter learning problem to a decoupled set of local statistical estimation problems at each node of the graph. This link between probabilistic inference and parameter learning is an important one, allowing developments in efficient inference to have immediate impact on research on learning algorithms.

The problem of learning the structure of a graph from data is significantly harder. In practice, most structure learning methods are heuristic methods that perform local search by starting with a given graph and improving it by adding or deleting one edge at a time (Heckerman et al., 1995; Cooper & Herskovits, 1992).

There is an important special case in which both parameter learning and structure learning are tractable, namely the case of graphical models in the form of a *tree distribution*. As shown by Chow and Liu (1968), the tree distribution that maximizes the likelihood of a set of observations on M nodes—as well as the parameters of the tree—can be found in time quadratic in the number of variables in the domain. This algorithm is known as the Chow-Liu algorithm.

Trees also have the virtue that probabilistic inference is guaranteed to be efficient, and indeed historically the earliest research in AI on efficient inference focused on trees (Pearl, 1988). Later research extended this early work by first considering general singly-connected graphs (Pearl, 1988), and then considering graphs with arbitrary (acylic) patterns of connectivity (Cowell et al., 1999). This line of research has provided one useful "upgrade path" from tree distributions to the complex Bayesian and Markov networks currently being studied.

In this paper we consider an alternative upgrade path. Inspired by the success of mixture models in providing simple, effective generalizations of classical methods in many simpler density estimation settings (MacLachlan & Bashford, 1988), we consider a generalization of tree distributions known as the *mixtures-of-trees (MT) model*. As suggested in Figure 1, the MT model involves the probabilistic mixture of a set of graphical components, each of which is a tree. In this paper we describe likelihood-based algorithms for learning the parameters and structure of such models.

One can also consider probabilistic mixtures of more general graphical models; indeed, the general case is the *Bayesian multinet* introduced by Geiger and Heckerman (1996). The Bayesian multinet is a mixture model in which each mixture component is an arbitrary graphical model. The advantage of Bayesian multinets over more traditional graphical mod-

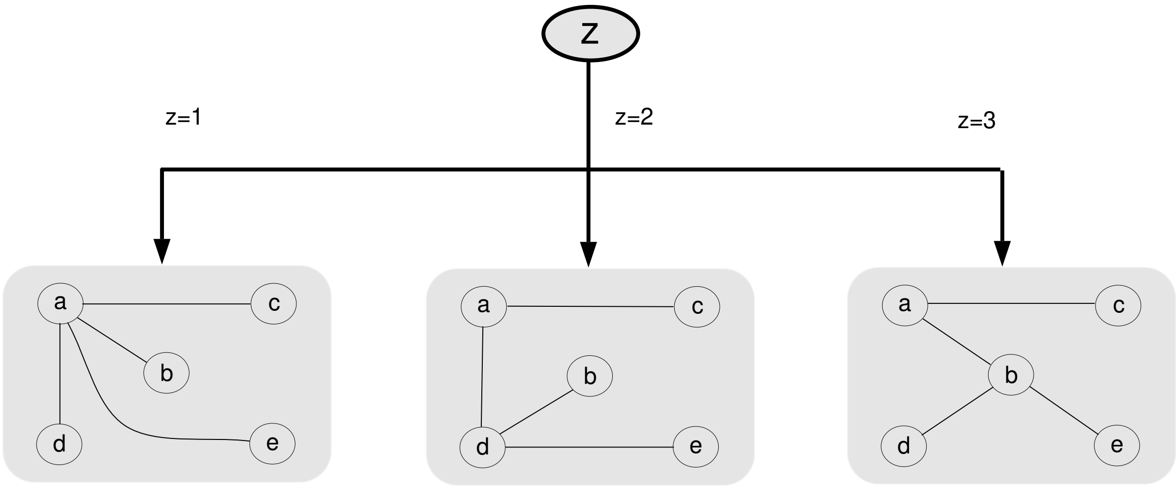


Figure 1: A mixture of trees over a domain consisting of random variables $V = \{a, b, c, d, e\}$, where z is a hidden *choice variable*. Conditional on the value of z, the dependency structure is a tree. A detailed presentation of the mixture-of-trees model is provided in Section 3.

els is the ability to represent *context-specific independencies*—situations in which subsets of variables exhibit certain conditional independencies for some, but not all, values of a conditioning variable. (Further work on context-specific independence has been presented by Boutilier, Friedman, Goldszmidt, & Koller, 1996). By making context-specific independencies explicit as multiple collections of edges, one can obtain (a) more parsimonious representations of joint probabilities and (b) more efficient inference algorithms.

In the machine learning setting, however, the advantages of the general Bayesian multinet formalism are less apparent. Allowing each mixture component to be a general graphical model forces us to face the difficulties of learning general graphical structure. Moreover, greedy edge-addition and edge-deletion algorithms seem particularly ill-suited to the Bayesian multinet, given that it is the focus on collections of edges rather than single edges that underlies much of the intuitive appeal of this architecture.

We view the mixture of trees as providing a reasonable compromise between the simplicity of tree distributions and the expressive power of the Bayesian multinet, while doing so within a restricted setting that leads to efficient machine learning algorithms. In particular, as we show in this paper, there is a simple generalization of the Chow-Liu algorithm that makes it possible to find (local) maxima of likelihoods (or penalized likelihoods) efficiently in general MT models. This algorithm is an iterative Expectation-Maximization (EM) algorithm, in which the inner loop (the M step) involves invoking the Chow-Liu algorithm to determine the structure and parameters of the individual mixture components. Thus, in a very concrete sense, this algorithm searches in the space of collections of edges.

In summary, the MT model is a multiple network representation that shares many of the basic features of Bayesian and Markov network representations, but brings new features to the fore. We believe that these features expand the scope of graph-theoretic probabilistic representations in useful ways and may be particularly appropriate for machine learning problems.

1.1 Related work

The MT model can be used both in the classification setting and the density estimation setting, and it makes contact with different strands of previous literature in these two guises.

In the classification setting, the MT model builds on the seminal work on tree-based classifiers by Chow and Liu (1968), and on recent extensions due to Friedman et al. (1997) and Friedman, Goldszmidt, and Lee (1998). Chow and Liu proposed to solve M-way classification problems by fitting a separate tree to the observed variables in each of the M classes, and classifying a new data point by choosing the class having maximum class-conditional probability under the corresponding tree model. Friedman et al. took as their point of departure the Naive Bayes model, which can be viewed as a graphical model in which an explicit class node has directed edges to an otherwise disconnected set of nodes representing the input variables (i.e., attributes). Introducing additional edges between the input variables yields the *Tree Augmented Naive Bayes (TANB) classifier* (Friedman et al., 1997; Geiger, 1992). These authors also considered a less constrained model in which different patterns of edges were allowed for each value of the class node—this is formally identical to the Chow and Liu proposal.

If the choice variable of the MT model is identified with the class label then the MT model is identical to the Chow and Liu approach (in the classification setting). However, we do not necessarily wish to identify the choice variable with the class label, and, indeed, in our experiments on classification we treat the class label as simply another input variable. This yields a more discriminative approach to classification in which all of the training data are pooled for the purposes of training the model (Section 5, Meilă & Jordan, 1998). The choice variable remains hidden, yielding a mixture model for each class. This is similar in spirit to the "mixture discriminant analysis" model of Hastie and Tibshirani (1996), where a mixture of Gaussians is used for each class in a multiway classification problem.

In the setting of density estimation, clustering and compression problems, the MT model makes contact with the large and active literature on mixture modeling. Let us briefly review some of the most salient connections. The Auto-Class model (Cheeseman & Stutz, 1995) is a mixture of factorial distributions (MF), and its excellent cost/performance ratio motivates the MT model in much the same way as the Naive Bayes model motivates the TANB model in the classification setting. (A *factorial* distribution is a product of factors each of which depends on exactly one variable). Kontkanen, Myllymaki, and Tirri (1996) study a MF in which a hidden variable is used for classification; this approach was extended by Monti and Cooper (1998). The idea of learning tractable but simple belief networks and superimposing a mixture to account for the remaining dependencies was developed independently of our work by Thiesson, Meek, Chickering, and Heckerman (1997), who studied mixtures of Gaussian belief networks. Their work interleaves EM parameter search with Bayesian model search in a heuristic but general algorithm.

2. Tree distributions

In this section we introduce the tree model and the notation that will be used throughout the paper. Let V denote a set of n discrete random variables of interest. For each random variable $v \in V$ let $\Omega(v)$ represent its range, $x_v \in \Omega(v)$ a particular value, and r_v the (finite) cardinality of $\Omega(v)$. For each subset A of V, let $\Omega(A) = \bigotimes_{v \in A} \Omega(v)$ and let x_A denote an assignment to the variables in A. To simplify notation x_V will be denoted by x and $\Omega(V)$ will be denoted simply by Ω. Sometimes we need to refer to the maximum of r_v over V; we denote this value by r_{max}.

We begin with undirected (Markov random field) representations of tree distributions. Identifying the vertex set of a graph with the set of random variables V, consider a graph $G = (V, E)$, where E is a set of undirected edges. We allow a tree to have multiple connected components (thus our "trees" are generally called *forests*). Given this definition, the number of edges $|E|$ and the number of connected components p are related as follows:

$$|E| + p \;=\; |V|,$$

implying that adding an edge to a tree reduces the number of connected components by 1. Thus, a tree can have at most $|V| - 1 = n - 1$ edges. In this latter case we refer to the tree as a *spanning tree*.

We parameterize a tree in the following way. For $u, v \in V$ and $(u, v) \in E$, let T_{uv} denote a joint probability distribution on u and v. We require these distributions to be consistent with respect to marginalization, denoting by $T_u(x_u)$ the marginal of $T_{uv}(x_u, x_v)$ or $T_{vu}(x_v, x_u)$ with respect to x_v for any $v \neq u$. We now assign a distribution T to the graph (V, E) as follows:

$$T(x) \;=\; \frac{\prod_{(u,v)\in E} T_{uv}(x_u, x_v)}{\prod_{v \in V} T_v(x_v)^{\deg v - 1}}, \tag{1}$$

where $\deg v$ is the *degree* of vertex v; i.e., the number of edges incident to $v \in V$. It can be verified that T is in fact a probability distribution; moreover, the pairwise probabilities T_{uv} are the marginals of T.

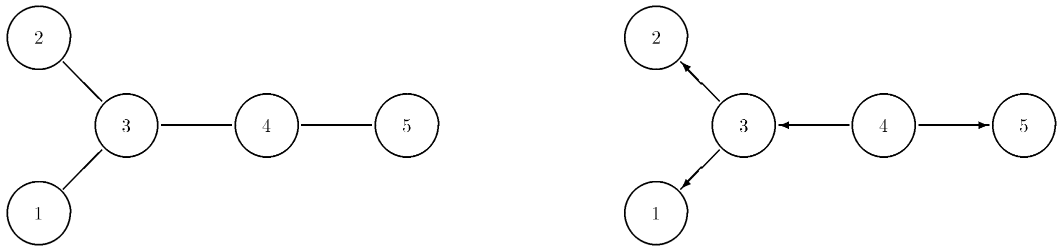
A *tree distribution* T is defined to be any distribution that admits a factorization of the form (1).

Tree distributions can also be represented using directed (Bayesian network) graphical models. Let $G^D = (V, E^D)$ denote a directed tree (possibly a forest), where E^D is a set of directed edges and where each node v has (at most) one parent, denoted $\mathrm{pa}(v)$. We parameterize this graph as follows:

$$T(x) \;=\; \prod_{v \in V} T_{v|\mathrm{pa}(v)}(x_v | x_{\mathrm{pa}(v)}) \tag{2}$$

where $T_{v|\mathrm{pa}(v)}(x_v | x_{\mathrm{pa}(v)})$ is an arbitrary conditional distribution. It can be verified that T indeed defines a probability distribution; moreover, the marginal conditionals of T are given by the conditionals $T_{v|\mathrm{pa}(v)}$.

We shall call the representations (1) and (2) the *undirected* and *directed* tree representations of the distribution T respectively. We can readily convert between these representations; for example, to convert (1) to a directed representation we choose an arbitrary root



$$T(x) = \frac{T_{13}(x_1x_3)\,T_{23}(x_2x_3)\,T_{34}(x_3x_4)\,T_{45}(x_4x_5)}{T_3^2(x_3)T_4(x_4)}$$

(a)

$$\begin{aligned} T(x) &= T_4(x_4)\frac{T_{34}(x_3x_4)}{T_4(x_4)}\frac{T_{54}(x_5x_4)}{T_4(x_4)}\frac{T_{13}(x_1x_3)}{T_3(x_3)}\frac{T_{23}(x_2x_3)}{T_3(x_3)} \\ &= T_4(x_4)T_{5|4}(x_5|x_4)T_{3|4}(x_3|x_4)T_{1|3}(x_1|x_3)T_{2|3}(x_2|x_3) \end{aligned}$$

(b)

Figure 2: A tree in its undirected (a) and directed (b) representations.

in each connected component and direct each edge away from the root. For $(u, v) \in E$ with u closer to the root than v, let $\mathrm{pa}(v) = u$. Now compute the conditional probabilities corresponding to each directed edge by recursively substituting $T_{v\mathrm{pa}(v)}/T_{\mathrm{pa}(v)}$ by $T_{v|\mathrm{pa}(v)}$ starting from the root. Figure 2 illustrates this process on a tree with 5 vertices.

The directed tree representation has the advantage of having independent parameters. The total number of free parameters in either representation is:

$$\sum_{(u,v)\in E} r_u r_v - \sum_{v\in V}(\deg v - 1)r_v - p \;=\; \sum_{(u,v)\in E}(r_u - 1)(r_v - 1) + \sum_{v\in V} r_v - n \qquad (3)$$

The right-hand side of (3) shows that each edge (u, v) increases the number of parameters by $(r_u - 1)(r_v - 1)$.

The set of conditional independencies associated with a tree distribution are readily characterized (Lauritzen, Dawid, Larsen, & Leimer, 1990). In particular, two subsets $A, B \subset V$ are independent given $C \subset V$ if C intersects every path (ignoring the direction of edges in the directed case) between u and v for all $u \in A$ and $v \in B$.

2.1 Marginalization, inference and sampling in tree distributions

The basic operations of computing likelihoods, conditioning, marginalization, sampling and inference can be performed efficiently in tree distributions; in particular, each of these operations has time complexity $\mathcal{O}(n)$. This is a direct consequence of the factorized representation of tree distributions in equations (1) and (2).

2.2 Representational capabilities

If graphical representations are natural for human intuition, then the subclass of tree models are particularly intuitive. Trees are sparse graphs, having $n - 1$ or fewer edges. There is at most one path between every pair of variables; thus, independence relationships between subsets of variables, which are not easy to read out in general Bayesian network topologies, are obvious in a tree. In a tree, an edge corresponds to the simple, common-sense notion of direct dependency and is the natural representation for it. However, the very simplicity that makes tree models appealing also limits their modeling power. Note that the number of free parameters in a tree grows linearly with n while the size of the state space $\Omega(V)$ is an exponential function of n. Thus the class of dependency structures representable by trees is a relatively small one.

2.3 Learning of tree distributions

The learning problem is formulated as follows: we are given a set of observations $\mathcal{D} = \{x^1, x^2, \ldots, x^N\}$ and we are required to find the tree T^* that maximizes the log likelihood of the data:

$$T^* \;=\; \operatorname*{argmax}_T \; \sum_{i=1}^{N} \log T(x^i),$$

where x^i is an assignment of values to all variables. Note that the maximum is taken *both* with respect to the tree structure (the choice of which edges to include) and with respect to the numerical values of the parameters. Here and in the rest of the paper we will assume for simplicity that there are no missing values for the variables in V, or, in other words, that the observations are *complete*.

Letting $P(x)$ denote the proportion of observations x_i in the training set $\mathcal{D}$ that are equal to x, we can alternatively express the maximum likelihood problem by summing over configurations x:

$$T^* \;=\; \operatorname*{argmax}_T \; \sum_{x \in \Omega} P(x) \log T(x).$$

In this form we see that the log likelihood criterion function is a (negative) cross-entropy. We will in fact solve the problem in general, letting $P(x)$ be an arbitrary probability distribution. This generality will prove useful in the following section where we consider mixtures of trees.

The solution to the learning problem is an algorithm, due to Chow and Liu (1968), that has quadratic complexity in n (see Figure 3). There are three steps to the algorithm. First, we compute the *pairwise marginals* $P_{uv}(x_u, x_v) = \sum_{x_{V-\{u,v\}}} P(x_u, x_v, x_{V-\{u,v\}})$. If P is an empirical distribution, as in the present case, computing these marginals requires $\mathcal{O}(n^2 N)$ operations. Second, from these marginals we compute the mutual information between each pair of variables in V under the distribution P:

$$I_{uv} \;=\; \sum_{x_u x_v} P_{uv}(x_u, x_v) \log \frac{P_{uv}(x_u, x_v)}{P_u(x_u) P_v(x_v)}, \qquad u, v \in V, u \neq v,$$

an operation that requires $\mathcal{O}(n^2 r_{MAX}^2)$ operations. Third, we run a maximum-weight spanning tree (MWST) algorithm (Cormen, Leiserson, & Rivest, 1990), using I_{uv} as the weight for edge $(u, v), forall u, v \in V$. Such algorithms, which run in time $\mathcal{O}(n^2)$, return a spanning tree that maximizes the total mutual information for edges included in the tree.

Chow and Liu showed that the maximum-weight spanning tree also maximizes the likelihood over tree distributions T, and moreover the optimizing parameters T^k_{uv} (or $T^k_{u|v}$), for $(u, v) \in E_{T^k}$, are equal to the corresponding marginals P_{uv} of the distribution P:

$$T_{uv} \;\equiv\; P_{uv}.$$

The algorithm thus attains a global optimum over both structure and parameters.

Algorithm ChowLiu(P)

Input: Distribution P over domain V
Procedure MWST(weights) that outputs a maximum weight spanning tree over V

1. Compute marginal distributions P_v, P_{uv} for $u, v \in V$
2. Compute mutual information values I_{uv} for $u, v \in V$
3. $E_T = \mathrm{MWST}(\{I_{uv}\})$
4. Set $T_{uv} \equiv P_{uv}$ for $uv \in E_T$

Output: T

Figure 3: The Chow and Liu algorithm for maximum likelihood estimation of tree structure and parameters.

3. Mixtures of trees

We define a mixture-of-trees (MT) model to be a distribution of the form:

$$Q(x) = \sum_{k=1}^{m} \lambda_k T^k(x)$$

with

$$\lambda_k \geq 0, \; k = 1, \ldots, m; \qquad \sum_{k=1}^{m} \lambda_k = 1.$$

The tree distributions T^k are the *mixture components* and λ_k are called *mixture coefficients*. A mixture of trees can be viewed as containing an unobserved *choice variable* z, which takes value $k \in \{1, \ldots m\}$ with probability λ_k. Conditioned on the value of z, the distribution of the observed variables V is a tree. The m trees may have different structures and different parameters. In Figure 1, for example, we have a mixture of trees with $m = 3$ and $n = 5$.

Note that because of the varying structure of the component trees, a mixture of trees is neither a Bayesian network nor a Markov random field. Let us adopt the notation

$$A \perp_P B \mid C$$

for "A is independent of B given C under distribution P". If for some (all) $k \in \{1, \ldots m\}$ we have

$$A \perp_{T^k} B \mid C \text{ with } A, B, C \subset V,$$

this will not imply that

$$A \perp_Q B \mid C.$$

On the other hand, a mixture of trees is capable of representing dependency structures that are conditioned on the value of a variable (the choice variable), something that a usual

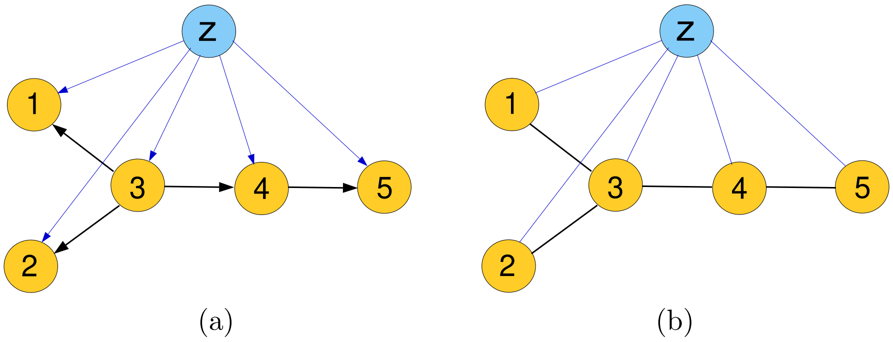


Figure 4: A mixture of trees with shared structure (MTSS) represented as a Bayes net (a) and as a Markov random field (b).

Bayesian network or Markov random field cannot do. Situations where such a model is potentially useful abound in real life: Consider for example bitmaps of handwritten digits. Such images obviously contain many dependencies between pixels; however, the pattern of these dependencies will vary across digits. Imagine a medical database recording the body weight and other data for each patient. The body weight could be a function of age and height for a healthy person, but it would depend on other conditions if the patient suffered from a disease or were an athlete. If, in a situation like the ones mentioned above, conditioning on one variable produces a dependency structure characterized by sparse, acyclic pairwise dependencies, then a mixture of trees may provide a good model of the domain.

If we constrain all of the trees in the mixture to have the same structure we obtain a *mixture of trees with shared structure* (MTSS; see Figure 4). In the case of the MTSS, if for some (all) $k \in \{1, \ldots m\}$

$$A \ \perp_{T^k} \ B \,|\, C,$$

then

$$A \ \perp_Q \ B \,|\, C \cup \{z\}.$$

In addition, a MTSS *can* be represented as a Bayesian network (Figure 4,a), as a Markov random field (Figure 4,b) and as a *chain graph* (Figure 5). Chain graphs were introduced by Lauritzen (1996); they represent a superclass of both Bayesian networks and Markov random fields. A chain graph contains both directed and undirected edges.

While we generally consider problems in which the choice variable is hidden (i.e., unobserved), it is also possible to utilize both the MT and the MTSS frameworks in which the choice variable is observed. Such models, which—as we discuss in Section 1.1—have been studied previously by Friedman et al. (1997) and Friedman et al. (1998), will be referred to generically as *mixtures with observed choice variable*. Unless stated otherwise, it will be assumed that the choice variable is hidden.

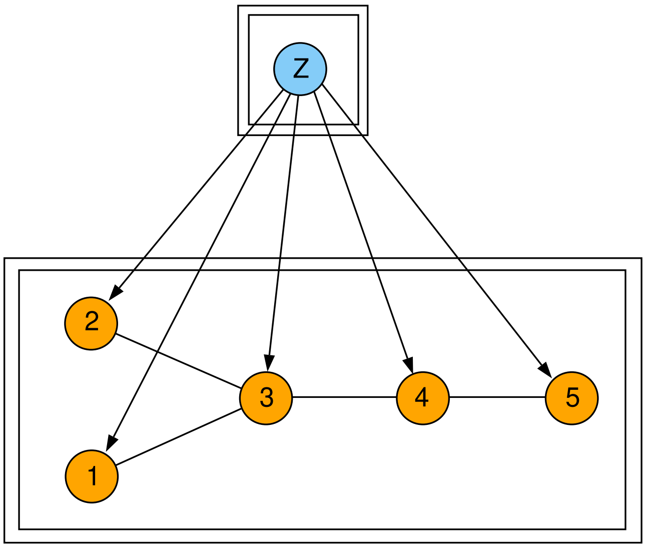


Figure 5: A MTSS represented as a chain graph. The double boxes enclose the undirected blocks of the chain graph.

3.1 Marginalization, inference and sampling in MT models

Let $Q(x) = \sum_{k=1}^{m} \lambda_k T^k(x)$ be an MT model. We consider the basic operations of marginalization, inference and sampling, recalling that these operations have time complexity $\mathcal{O}(n)$ for each of the component tree distributions T^k.

Marginalization. The marginal distribution of a subset $A \subset V$ is given as follows:

$$Q_A(x_A) = \sum_{k=1}^{m} \lambda_k T_A^k(x_A).$$

Hence, the marginal of Q is a mixture of the marginals of the component trees.

Inference. Let $V' = x_{V'}$ be the evidence. Then the probability of the hidden variable given evidence $V' = x_{V'}$ is obtained by applying Bayes' rule as follows:

$$Pr[z = k | V' = x_{V'}] = \frac{\lambda_k T_{V'}^k(V' = x_{V'})}{\sum_{k'} \lambda_{k'} T_{V'}^{k'}(V' = x_{V'})}.$$

In particular, when we observe all but the choice variable, i.e., $V' \equiv V$ and $x_{V'} \equiv x$, we obtain the *posterior probability distribution* of z:

$$Pr[z = k | V = x] \equiv Pr[z = k | x] = \frac{\lambda_k T^k(x)}{\sum_{k'} \lambda_{k'} T^{k'}(x)}. \quad (4)$$

The probability distribution of a given subset of V given the evidence is

$$Q_{A|V'}(x_A | x_{V'}) = \frac{\sum_{k=1}^{m} \lambda_k T_{A,V'}^k(x_A, x_{V'})}{\sum_{k=1}^{m} \lambda_k T_{V'}^k(x_{V'})} = \sum_{k=1}^{m} Pr[z = k | V' = x_{V'}] \, T_{A|V'}^k(x_A | x_{V'}).$$

Thus the result is again a mixture of the results of inference procedures run on the component trees.

Sampling. The procedure for sampling from a mixture of trees is a two stage process: first one samples a value k for the choice variable z from its distribution $(\lambda_1, \lambda_2, \ldots \lambda_m)$, then a value x is sampled from T^k using the procedure for sampling from a tree distribution.

In summary, the basic operations on mixtures of trees, marginalization, conditioning and sampling, are achieved by performing the corresponding operation on each component of the mixture and then combining the results. Therefore, the complexity of these operations scales linearly with the number of trees in the mixture.

3.2 Learning of MT models

The expectation-maximization (EM) algorithm provides an effective approach to solving many learning problems (Dempster, Laird, & Rubin, 1977; MacLachlan & Bashford, 1988), and has been employed with particular success in the setting of mixture models and more general latent variable models (Jordan & Jacobs, 1994; Jelinek, 1997; Rubin & Thayer, 1983). In this section we show that EM also provides a natural approach to the learning problem for the MT model.

An important feature of the solution that we present is that it provides estimates for both parametric and structural aspects of the model. In particular, although we assume (in the current section) that the number of trees is fixed, the algorithm that we derive provides estimates for both the pattern of edges of the individual trees and their parameters. These estimates are maximum likelihood estimates, and although they are subject to concerns about overfitting, the constrained nature of tree distributions helps to ameliorate overfitting problems. It is also possible to control the number of edges indirectly via priors; we discuss methods for doing this in Section 4.

We are given a set of observations $\mathcal{D} = \{x^1, x^2, \ldots, x^N\}$ and are required to find the mixture of trees Q that satisfies

$$Q = \operatorname*{argmax}_{Q'} \sum_{i=1}^{N} \log Q(x^i).$$

Within the framework of the EM algorithm, this likelihood function is referred to as the *incomplete log-likelihood* and it is contrasted with the following *complete log-likelihood* function:

$$\begin{aligned} l_c(x^{1,\ldots N}, z^{1,\ldots N}|Q) &= \sum_{i=1}^{N} \log \prod_{k=1}^{m} (\lambda_k T^k(x^i))^{\delta_{k,z^i}} \\ &= \sum_{i=1}^{N} \sum_{k=1}^{m} \delta_{k,z^i} (\log \lambda_k + \log T^k(x^i)), \end{aligned} \tag{5}$$

where δ_{k,z^i} is equal to one if z^i is equal to the kth value of the choice variable and zero otherwise. The complete log-likelihood would be the log-likelihood of the data if the *unobserved data* $\{z^1, z^2, \ldots, z^N\}$ could be observed.

The idea of the EM algorithm is to utilize the complete log-likelihood, which is generally easy to maximize, as a surrogate for the incomplete log-likelihood, which is generally somewhat less easy to maximize directly. In particular, the algorithm goes uphill in the expected value of the complete log-likelihood, where the expectation is taken with respect to the unobserved data. The algorithm thus has the form of an interacting pair of steps: the *E step*, in which the expectation is computed given the current value of the parameters, and the *M step*, in which the parameters are adjusted so as to maximize the expected complete log-likelihood. These two steps iterate and are proved to converge to a local maximum of the (incomplete) log-likelihood (Dempster et al., 1977).

Taking the expectation of (5), we see that the E step for the MT model reduces to taking the expectation of the delta function δ_{k,z^i}, conditioned on the data $\mathcal{D}$:

$$E[\delta_{k,z^i}|\mathcal{D}] = Pr[z^i = k|\mathcal{D}] = Pr[z^i = k|V = x^i],$$

and this latter quantity is recognizable as the posterior probability of the hidden variable given the ith observation (cf. equation (4)). Let us define:

$$\gamma_k(i) \;=\; \frac{\lambda_k T^k(x^i)}{\sum_{k'} \lambda_{k'} T^{k'}(x^i)} \tag{6}$$

as this posterior probability.

Substituting (6) into the expected value of the complete log-likelihood in (5), we obtain:

$$E[l_c(x^{1,\dots N}, z^{1,\dots N}|Q)] \;=\; \sum_{i=1}^{N}\sum_{k=1}^{m} \gamma_k(i)(\log\lambda_k + \log T^k(x^i)).$$

Let us define the following quantities:

$$\begin{aligned}\Gamma_k &= \sum_{i=1}^{N} \gamma_k(x^i), \quad k = 1, \dots m\\ P^k(x^i) &= \frac{\gamma_k(i)}{\Gamma_k},\end{aligned}$$

where the sums $\Gamma_k \in [0, N]$ can be interpreted as the total number of data points that are generated by component T^k. Using these definitions we obtain:

$$E[l_c(x^{1,\dots N}, z^{1,\dots N}\,|\,Q)] \;=\; \sum_{k=1}^{m} \Gamma_k \log\lambda_k + \sum_{k=1}^{m} \Gamma_k \sum_{i=1}^{N} P^k(x^i) \log T^k(x^i). \tag{7}$$

It is this quantity that we must maximize with respect to the parameters.

From (7) we see that $E[l_c]$ separates into terms that depend on disjoint subsets of the model parameters and thus the M step decouples into separate maximizations for each of the various parameters. Maximizing the first term of (7) with respect to the parameters λ, subject to the constraint $\sum_{k=1}^{m} \lambda_k \;=\; 1$, we obtain the following update equation:

$$\lambda_k \;=\; \frac{\Gamma_k}{N} \text{ for } k = 1, \dots m.$$

Algorithm MIXTREE($\mathcal{D}, Q_0$)

Input: Dataset $\{x^1, \dots x^N\}$
Initial model $Q_0 = \{m,\ T^k,\ \lambda^k,\ k = 1, \dots m\}$
Procedure CHOWLIU(P)

Iterate until convergence:
E step: Compute $\gamma_k^i,\ P^k(x^i)$ for $k = 1, \dots m,\ i = 1, \dots N$
M step: for $k = 1, \dots m$
$\lambda_k \leftarrow \Gamma_k / N$
$T^k\ = \text{CHOWLIU}(P^k)$

Output: Model $Q = \{m,\ T^k,\ \lambda^k,\ k = 1, \dots m\}$

Figure 6: The MIXTREE algorithm for learning MT models.

In order to update T^k, we see that we must maximize the negative cross-entropy between P^k and T^k:

$$\sum_{i=1}^{N} P^k(x^i) \log T^k(x^i).$$

This problem is solved by the CHOWLIU algorithm from Section 2.3. Thus we see that the M step for learning the mixture components of MT models reduces to m separate runs of the CHOWLIU algorithm, where the "target" distribution $P^k(x^i)$ is the normalized posterior probability obtained from the E step.

We summarize the results of the derivation of the EM algorithm for mixtures of trees in Figure 6.

3.2.1 Running time

Computing the likelihood of a data point under a tree distribution (in the directed tree representation) takes $n - 1 = \mathcal{O}(n)$ multiplications. Hence, the E step requires $\mathcal{O}(mnN)$ floating point operations.

As for the M step, the most computationally expensive phase is the computation of the marginals P_{uv}^k for the kth tree. This step has time complexity $\mathcal{O}(mn^2N)$. Another $\mathcal{O}(mn^2r_{MAX}^2)$ and $\mathcal{O}(mn^2)$ are required at each iteration for the mutual informations and for the m runs of the MWST algorithm. Finally we need $\mathcal{O}(mnr_{MAX}^2)$ for computing the tree parameters in the directed representation. The total running time per EM iteration is thus

$$\mathcal{O}(mn^2N + mn^2r_{MAX}^2).$$

The algorithm is polynomial (per iteration) in the dimension of the domain, the number of components and the size of the data set.

The space complexity is also polynomial, and is dominated by $\mathcal{O}(n^2 r_{MAX}^2)$, the space needed to store the pairwise marginal tables P_{uv}^k (the tables can be overwritten by successive values of k).

3.3 Learning mixtures of trees with shared structure

It is possible to modify the MixTree algorithm so as to constrain the m trees to share the same structure, and thereby estimate MTSS models.

The E step remains unchanged. The only novelty is the reestimation of the tree distributions T^k in the M step, since they are now constrained to have the same structure. Thus, the maximization cannot be decoupled into m separate tree estimations but, remarkably enough, it can still be performed efficiently.

It can be readily verified that for any given structure the optimal parameters of each tree edge T_{uv}^k are equal to the parameters of the corresponding marginal distribution P_{uv}^k. It remains only to find the optimal structure. The expression to be optimized is the second sum in the right-hand side of equation (7). By replacing T_{uv}^k with P_{uv}^k and denoting the mutual information between u and v under P^k by I_{uv}^k this sum can be reexpressed as follows:

$$\begin{aligned}\sum_{k=1}^{m}\Gamma_k\sum_{i=1}^{N}P^k(x^i)\log T^k(x^i) &= N\sum_{k=1}^{m}\lambda_k[\sum_{(u,v)\in E} I_{uv}^k - \sum_{v\in V} H(P_v^k)] \\ &= N\sum_{(u,v)\in E} I_{uv|z} - N\sum_{v\in V} H(v|z). \end{aligned} \tag{8}$$

The new quantity $I_{uv|z}$ appearing above represents the mutual information of u and v conditioned on the hidden variable z. Its general definition for three discrete variables u, v, z distributed according to $P_{uvz} \equiv P$ is

$$I_{uv|z} = \sum_{x_z} P_z(x_z)\sum_{x_u x_v} P_{uv|z}(x_u x_v|x_z)\log\frac{P_{uv|z}(x_u x_v|x_z)}{P_{u|z}(x_u|x_z)P_{v|z}(x_v|x_z)}.$$

The second term in (8), $N\sum_{v\in V} H(v|z)$, represents the sum of the conditional entropies of the variables given z and is independent of the tree structure. Hence, the optimization of the structure E can be achieved by running a MWST algorithm with the edge weights represented by $I_{uv|z}$. We summarize the algorithm in Figure 7.

4. Decomposable priors and MAP estimation for mixtures of trees

The Bayesian learning framework combines information obtained from direct observations with prior knowledge about the model, when the latter is represented as a probability distribution. The object of interest of Bayesian analysis is the posterior distribution over the models given the observed data, $Pr[Q|\mathcal{D}]$, a quantity which can rarely be calculated explicitly. Practical methods for approximating the posterior include choosing a single *maximum a posteriori (MAP)* estimate, replacing the continuous space of models by a finite set $\mathcal{Q}$ of high posterior probability (Heckerman et al., 1995), and expanding the posterior around its mode(s) (Cheeseman & Stutz, 1995).

Algorithm MixTreeSS($\mathcal{D}, Q_0$)

Input: Dataset $\{x^1, \ldots x^N\}$
Initial model $Q_0 = \{m,\ T^k,\ \lambda^k,\ k = 1, \ldots m\}$
Procedure MWST(weights) that outputs a maximum weight spanning tree

Iterate until convergence:
E step: Compute $\gamma_k^i,\ P^k(x^i)$ for $k = 1, \ldots m,\ i = 1, \ldots N$
M step: Compute marginals $P_v^k,\ P_{uv}^k,\ u, v \in V$
Compute mutual information values $I_{uv|z}\ u, v \in V$
$E_T\ = \text{MWST}(\{\, I_{uv|z}\,\})$
Set $\lambda_k \leftarrow \Gamma_k/N$
Set $T_{uv}^k \equiv P_{uv}^k$ for $uv\ \in\ E_T$ and for $k = 1, \ldots m$

Output: Model $Q = \{m,\ T^k,\ \lambda^k,\ k = 1, \ldots m\}$

Figure 7: The MixTreeSS algorithm for learning MTSS models.

Finding the local maxima (modes) of the distribution $Pr[Q|\mathcal{D}]$ is a necessary step in all the above methods and is our primary concern in this section. We demonstrate that maximum a posteriori modes can be found as efficiently as maximum likelihood modes, given a particular choice of prior. This has two consequences: First, it makes approximate Bayesian averaging possible. Second, if one uses a *non-informative* prior, then MAP estimation is equivalent to Bayesian smoothing, and represents a form of regularization. Regularization is particularly useful in the case of small data sets in order to prevent overfitting.

4.1 MAP estimation by the EM algorithm

For a model Q and dataset $\mathcal{D}$ the logarithm of the posterior $Pr[Q|\mathcal{D}]$ equals:

$$\log Pr[Q] \ + \sum_{x \in \mathcal{D}} \log Q(x)$$

plus an additive constant.

The EM algorithm can be adapted to maximize the log posterior for every fixed m (Neal & Hinton, 1999). Indeed, by comparing with equation (5) we see that the quantity to be maximized is now:

$$E[\log Pr[Q|x^{1,\ldots,N}, z^{1,\ldots,N}]] \ = \ \log Pr[Q] \ + \ E[l_c(x^{1,\ldots,N}, z^{1,\ldots,N}|Q)]. \tag{9}$$

The prior term does not influence the E step of the EM algorithm, which proceeds exactly as before (cf. equation (6)). To be able to successfully maximize the right-hand side of (9) in the M step we require that log $Pr[Q]$ decomposes into a sum of independent terms matching the decomposition of $E[l_c]$ in (7). A prior over mixtures of trees that is amenable to this decomposition is called a *decomposable prior*. It will have the following product form

$$\begin{aligned} Pr[Q] &= Pr[\lambda_{1,\ldots,m}] \prod_{k=1}^{m} \underbrace{Pr[E_k] Pr[\text{parameters}|E_k]}_{Pr[T^k]} \\ &= Pr[\lambda_{1,\ldots,m}] \prod_{k=1}^{m} \left[\left(\prod_{(u,v)\in E_k} e^{-\beta_{uv}} \right) \left(\prod_{v\in V} Pr[T^k_{v|\mathrm{pa}(v)}] \right) \right]. \end{aligned}$$

The first parenthesized factor in this equation represents the prior of the tree structure E_k while the second factor is the prior for parameters.

Requiring that the prior be decomposable is equivalent to making several independence assumptions: in particular, it means that the parameters of each tree in the mixture are independent of the parameters of all the other trees as well as of the probability of the mixture variable. In the following section, we show that these assumptions are not overly restrictive, by constructing decomposable priors for tree structures and parameters and showing that this class is rich enough to contain members that are of practical importance.

4.2 Decomposable priors for tree structures

The general form of a decomposable prior for the tree structure E is one where each edge contributes a constant factor independent of the presence or absence of other edges in E:

$$Pr[E] \propto \prod_{(u,v)\in E} \exp(-\beta_{uv}).$$

With this prior, the expression to be maximized in the M step of the EM algorithm becomes

$$\sum_{k=1}^{m} \Gamma_k \log \lambda_k + \sum_{k=1}^{m} \Gamma_k \left(\sum_{i=1}^{N} P^k(x^i) \log T^k(x^i) - \sum_{(u,v)\in E_k} \frac{\beta_{uv}}{\Gamma_k} \right).$$

Consequently, each edge weight W^k_{uv} in tree T_k is adjusted by the corresponding value β_{uv} divided by the total number of points that tree k is responsible for:

$$W^k_{uv} = I^k_{uv} - \frac{\beta_{uv}}{\Gamma_k}.$$

A negative β_{uv} increases the probability of (u,v) being present in the final solution, whereas a positive value of β_{uv} acts like a penalty on the presence of edge (u,v) in the tree. If the MWST procedure is modified so as to not add negative-weight edges, one can obtain (disconnected) trees having fewer than $n-1$ edges. Note that the strength of the prior is inversely proportional to Γ_k, the total number of data points assigned to mixture component k. Thus, with equal priors for all trees T^k, trees accounting for fewer data points will be penalized more strongly and therefore will be likely to have fewer edges.

If one chooses the edge penalties to be proportional to the increase in the number of parameters caused by the addition of edge uv to the tree,

$$\beta_{uv} = \frac{1}{2}(r_u - 1)(r_v - 1) \log N$$

then a *Minimum Description Length* (MDL) (Rissanen, 1989) type of prior is implemented.

In the context of learning Bayesian networks, Heckerman et al. (1995) suggested the following prior:

$$Pr[E] \;\propto\; \kappa^{\Delta(E,E^*)}$$

where $\Delta(\cdot)$ is a distance metric between Bayes net structures and E^* is the prior network structure. Thus, this prior penalizes deviations from the prior network. This prior is decomposable, entailing

$$\beta_{uv} \;=\; \begin{cases} -\ln\kappa, & (u,v)\in E^* \\ \ln\kappa, & (u,v)\notin E^* \end{cases} .$$

Decomposable priors on structure can also be used when the structure is common for all trees (MTSS). In this case the effect of the prior is to penalize the weight $I_{uv|z}$ in (8) by $-\beta_{uv}/N$.

A decomposable prior has the remarkable property that its normalization constant can be computed exactly in closed form. This makes it possible not only to completely define the prior, but also to compute averages under this prior (e.g., to compute a model's evidence). Given that the number of all undirected tree structures over n variables is n^{n-2}, this result (Meilă & Jaakkola, 2000) is quite surprising.

4.3 Decomposable priors for tree parameters

The decomposable prior for parameters that we introduce is a *Dirichlet prior* (Heckerman et al., 1995). The Dirichlet distribution is defined over the domain of $\theta_{1,\ldots,r} > 0,\ \sum_j \theta_j = 1$ and has the form

$$D(\theta_{1,\ldots,r}; N'_{1,\ldots,r}) \;\propto\; \prod_{j=1}^{r} \theta_j^{N'_j - 1}.$$

The numbers $N'_{1,\ldots,r} > 0$ that parametrize D can be interpreted as the sufficient statistics of a "fictitious data set" of size $N' = \sum_j N'_j$. Therefore N'_j are called *fictitious counts.* N' represents the strength of the prior.

To specify a prior for tree parameters, one must specify a Dirichlet distribution for each of the probability tables $T_{v|u=x_u}, \overline{uv} \in E^D$, for each possible tree structure E^D. This is achieved by means of a set of parameters $N'_{vux_vx_u}$ satisfying

$$\sum_{x_u} N'_{vux_vx_u} \;=\; N'_{vx_v}, \qquad \sum_{x_v} N'_{vx_v} \;=\; N' \qquad \text{for all } u,v \in V.$$

With these settings, the prior for the parameters $T_{v|u}(x_v|x_u)$, $x_v = 1,\ldots,r_v$ in any tree that contains the directed edge $\overline{uv}$ is defined by $N'_{uvx_ux_v}$, $x_v = 1,\ldots,r_v$. This representation of the prior is not only compact (order $n^2 r_{MAX}^2$ parameters) but it is also consistent: two different directed parametrizations of the same tree distribution receive the same prior. The assumptions allowing us to define this prior are explicated by Meilă and Jaakkola (2000) and parallel the reasoning of Heckerman et al. (1995) for general Bayes nets.

Denote by P the empirical distribution obtained from a data set of size N and by $P'_{uv}(x_ux_v) = N'_{uvx_ux_v}/N'$ the distribution defined by the fictitious counts. Then, by a property of the Dirichlet distribution (Heckerman et al., 1995) it follows that learning a

MAP tree is equivalent to learning an ML tree for the weighted combination $\tilde{P}$ of the two "datasets"

$$\tilde{P} \;=\; \frac{1}{N+N'}(N'P' + NP). \tag{10}$$

Consequently, the parameters of the optimal tree will be $T_{uv} \;=\; \tilde{P}_{uv}$.

For a mixture of trees, maximizing the posterior translates into replacing P by P^k and N by Γ_k in equation (10) above. This implies that the M step of the EM algorithm, as well as the E step, is exact and tractable in the case of MAP estimation with decomposable priors.

Finally, note that the posteriors $Pr[Q|\mathcal{D}]$ for models with different m are defined up to a constant that depends on m. Hence, one cannot compare posteriors of MTs with different numbers of mixture components m. In the experiments that we present, we chose m via other performance criteria: validation set likelihood in the density estimation experiments and validation set classification accuracy in the classification tasks.

5. Experiments

This section describes the experiments that were run in order to assess the promise of the MT model. The first experiments are *structure identification* experiments; they examine the ability of the MixTree algorithm to recover the original distribution when the data are generated by a mixture of trees. The next group of experiments studies the performance of the MT model as a *density estimator*; the data used in these experiments are not generated by mixtures of trees. Finally, we perform *classification* experiments, studying both the MT model and a single tree model. Comparisons are made with classifiers trained in both supervised and unsupervised mode. The section ends with a discussion of the single tree classifier and its *feature selection* properties.

In all of the experiments the training algorithm is initialized at random, independently of the data. Unless stated otherwise, the learning algorithm is run until convergence. Log-likelihoods are expressed in bits/example and therefore are sometimes called *compression rates*. The lower the value of the compression rate, the better the fit to the data.

In the experiments that involve small data sets we use the Bayesian methods that we discussed in Section 4 to impose a penalty on complex models. In order to regularize model structure we use a decomposable prior over tree edges with $\beta_{uv} = \beta > 0$. To regularize model parameters we use a Dirichlet prior derived from the pairwise marginal distributions for the data set. This approach is known as *smoothing with the marginal* (Friedman et al., 1997; Ney, Essen, & Kneser, 1994). In particular, we set the parameter N'_k characterizing the Dirichlet prior for tree k by apportioning a fixed *smoothing coefficient* α equally between the n variables and in an amount that is inversely proportional to Γ_k between the m mixture components. Intuitively, the effect of this operation is to make the m trees more similar to each other, thereby reducing the effective model complexity.

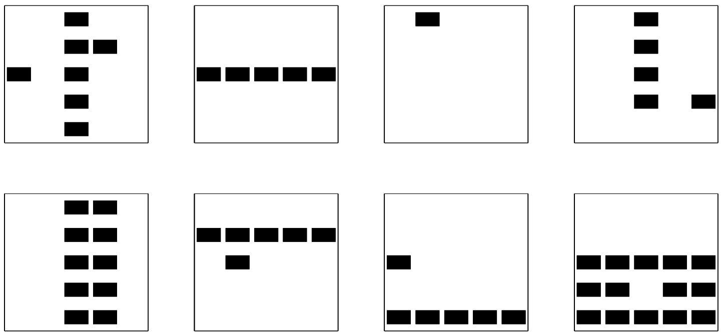


Figure 8: Eight training examples for the bars learning task.

5.1 Structure identification

5.1.1 Random trees, large data set

For the first structure identification experiment, we generated a mixture of $m = 5$ trees over 30 variables with each vertex having $r = 4$ values. The distribution of the choice variable λ as well as each tree's structure and parameters were sampled at random. The mixture was used to generate 30,000 data points that were used as a training set for a MixTree algorithm. The initial model had $m = 5$ components but otherwise was random. We compared the structure of the learned model with the generative model and computed the likelihoods of both the learned and the original model on a test dataset consisting of 1000 points.

The algorithm was quite successful at identifying the original trees: out of 10 trials, the algorithm failed to identify correctly only 1 tree in 1 trial. Moreover, this result can be accounted for by sampling noise; the tree that wasn't identified had a mixture coefficient λ of only 0.02. The difference between the log likelihood of the samples of the generating model and the approximating model was 0.41 bits per example.

5.1.2 Random bars, small data set

The "bars" problem is a benchmark structure learning problem for unsupervised learning algorithms in the neural network literature (Dayan & Zemel, 1995). The domain V is the $l \times l$ square of binary variables depicted in Figure 8. The data are generated in the following manner: first, one flips a fair coin to decide whether to generate horizontal or vertical bars; this represents the hidden variable in our model. Then, each of the l bars is turned on independently (black in Figure 8) with probability p_b. Finally, noise is added by flipping each bit of the image independently with probability p_n. A learner is shown data generated by this process; the task of the learner is to discover the data generating mechanism.

A mixture of trees model that approximates the true structure for low noise levels is shown in Figure 10. Note that any tree over the variables forming a bar is an equally good approximation. Thus, we will consider that the structure has been discovered when the model learns a mixture with $m = 2$, each T^k having l connected components, one for each bar. Additionally, we shall test the classification accuracy of the learned model by

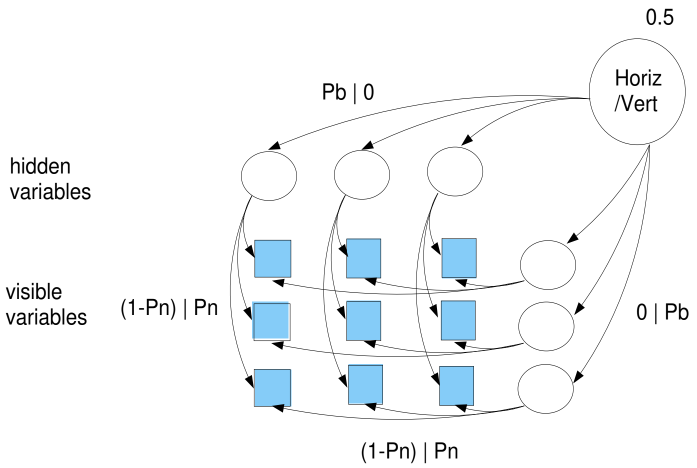


Figure 9: The true structure of the probabilistic generative model for the bars data.

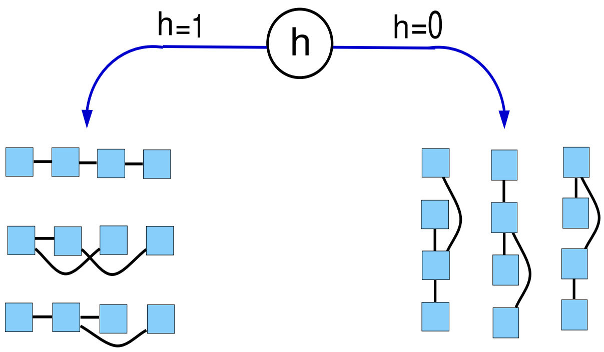


Figure 10: A mixture of trees that approximates the generative model for the bars problem. The interconnection between the variables in each "bar" are arbitrary.

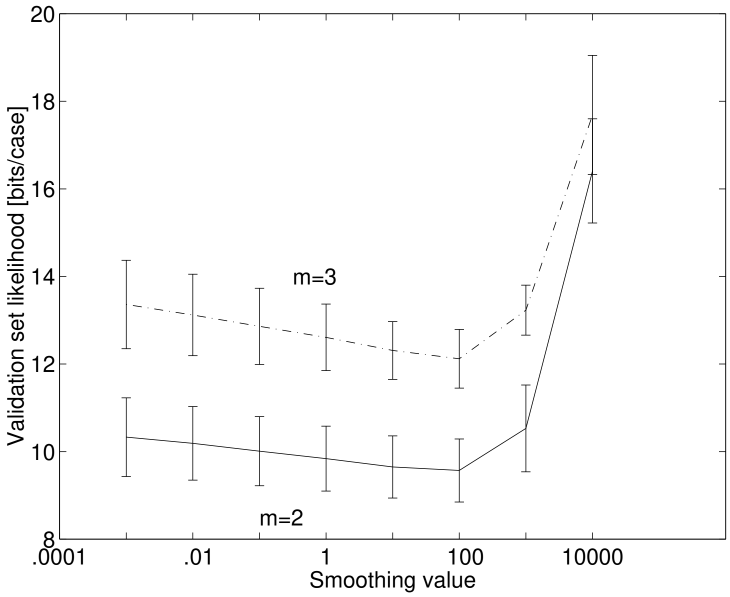


Figure 11: Test set log-likelihood on the bars learning task for different values of the smoothing parameter α and different m. The figure presents averages and standard deviations over 20 trials.

comparing the true value of the hidden variable (i.e. "horizontal" or "vertical") with the value estimated by the model for each data point in a test set.

As seen in the first row, third column of Figure 8, some training set examples are ambiguous. We retained these ambiguous examples in the training set. The total training set size was $N_{train} = 400$. We trained models with $m = 2, 3, \ldots$, and evaluated the models on a validation set of size 100 to choose the final values of m and of the smoothing parameter α. Typical values for l in the literature are $l = 4, 5$; we choose $l = 5$ following Dayan and Zemel (1995). The other parameter values were $p_b = 0.2, p_n = 0.02$ and $N_{test} = 200$. To obtain trees with several connected components we used a small edge penalty $\beta = 5$.

The validation-set log-likelihoods (in bits) for $m = 2, 3$ are given in Figure 11. Clearly, $m = 2$ is the best model. For $m = 2$ we examined the resulting structures: in 19 out of 20 trials, structure recovery was perfect. Interestingly, this result held for the whole range of the smoothing parameter α, not simply at the cross-validated value. By way of comparison, Dayan and Zemel (1995) examined two training methods and the structure was recovered in 27 and respectively 69 cases out of 100.

The ability of the learned representation to categorize new examples as coming from one group or the other is referred to as classification performance and is shown in Table 1. The result reported is obtained on a separate test set for the final, cross-validated value of α. Note that, due to the presence of ambiguous examples, no model can achieve perfect classification. The probability of an ambiguous example is $p_{ambig} = p_b^l + (1-p_b)^l \approx 0.25$, which yields an error rate of $0.5p_{ambig} = 0.125$. Comparing this lower bound with the value

Table 1: Results on the bars learning task.

Test set	ambiguous	unambiguous
l_{test} [bits/datapt]	9.82 ± 0.95	13.67 ± 0.60
Class accuracy	0.852 ± 0.076	0.951 ± 0.006

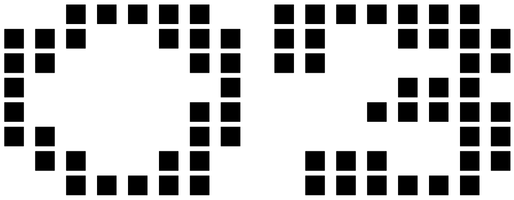


Figure 12: An example of a digit pair.

in the corresponding column of Table 1 shows that the model performs quite well, even when trained on ambiguous examples.

To further support this conclusion, a second test set of size 200 was generated, this time including only non-ambiguous examples. The classification performance, shown in the corresponding section of Table 1, rose to 0.95. The table also shows the likelihood of the (test) data evaluated on the learned model. For the first, "ambiguous" test set, this is 9.82, 1.67 bits away from the true model likelihood of 8.15 bits/data point. For the "non-ambiguous" test set, the compression rate is significantly worse, which is not surprising given that the distribution of the test set is now different from the distribution the model was trained on.

5.2 Density estimation experiments

In this section we present the results of three experiments that study the mixture of trees in the density estimation setting.

5.2.1 Digits and digit pairs images

Our first density estimation experiment involved a subset of binary vector representations of handwritten digits. The datasets consist of normalized and quantized 8×8 binary images of handwritten digits made available by the US Postal Service Office for Advanced Technology. One dataset—which we refer to as the "digits" dataset—contains images of single digits in 64 dimensions. The other dataset ("pairs") contains 128-dimensional vectors representing randomly paired digit images. These datasets, as well as the training conditions that we employed, are described by Frey, Hinton, and Dayan (1996). (See Figure 12 for an example of a digit pair). The training, validation and test sets contained 6000, 2000, and 5000 exemplars respectively. Each model was trained on the training set until the likelihood of the validation set stopped increasing.

We tried mixtures of 16, 32, 64 and 128 trees, fit by the MixTree algorithm. For each of the digits and pairs datasets we chose the mixture model with the highest log-likelihood

Table 2: Average log-likelihood (bits per digit) for the single digit (Digit) and double digit (Pairs) datasets. Results are averaged over 3 runs.

m	Digits	Pairs
16	34.72	79.25
32	34.48	78.99
64	34.84	79.70
128	34.88	81.26

on the validation set and using it we calculated the average log-likelihood over the test set (in bits per example). The averages (over 3 runs) are shown in Table 2.

In Figure 13 we compare our results (for $m = 32$) with the results published by Frey et al. (1996). The algorithms plotted in the figure are the (1) completely factored or "Base rate" (BR) model, which assumes that every variable is independent of all the others, (2) mixture of factorial distributions (MF), (3) the UNIX "gzip" compression program, (4) the Helmholtz Machine, trained by the wake-sleep algorithm (Frey et al., 1996) (HWS), (5) the same Helmholtz Machine where a mean field approximation was used for training (HMF), (5) a fully observed and fully connected sigmoid belief network (FV), and (6) the mixture of trees (MT) model.

As shown in Figure 13, the MT model yields the best density model for the simple digits and the second-best model for pairs of digits. A comparison of particular interest is between the MT model and the mixture of factorial (MF) model. In spite of the structural similarities in these models, the MT model performs better than the MF model, indicating that there is structure in the data that is exploited by the mixture of spanning trees but is not captured by a mixture of independent variable models. Comparing the values of the average likelihood in the MT model for digits and pairs we see that the second is more than twice the first. This suggests that our model (and the MF model) is able to perform good compression of the digit data but is unable to discover the independence in the double digit set.

5.2.2 The ALARM network

Our second set of density estimation experiments features the ALARM network as the data generating mechanism (Heckerman et al., 1995; Cheng, Bell, & Liu, 1997). This Bayesian network was constructed from expert knowledge as a medical diagnostic alarm message system for patient monitoring. The domain has $n = 37$ discrete variables taking between 2 and 4 values, connected by 46 directed arcs. Note that this network is *not* a tree or a mixture of trees, but the topology of the graph is sparse, suggesting the possibility of approximating the dependency structure by a mixture of trees with a small number of components m.

We generated a training set having $N_{train} = 10,000$ data points and a separate test set of $N_{test} = 2,000$ data points. On these sets we trained and compared the following methods: mixtures of trees (MT), mixtures of factorial (MF) distributions, the true model,

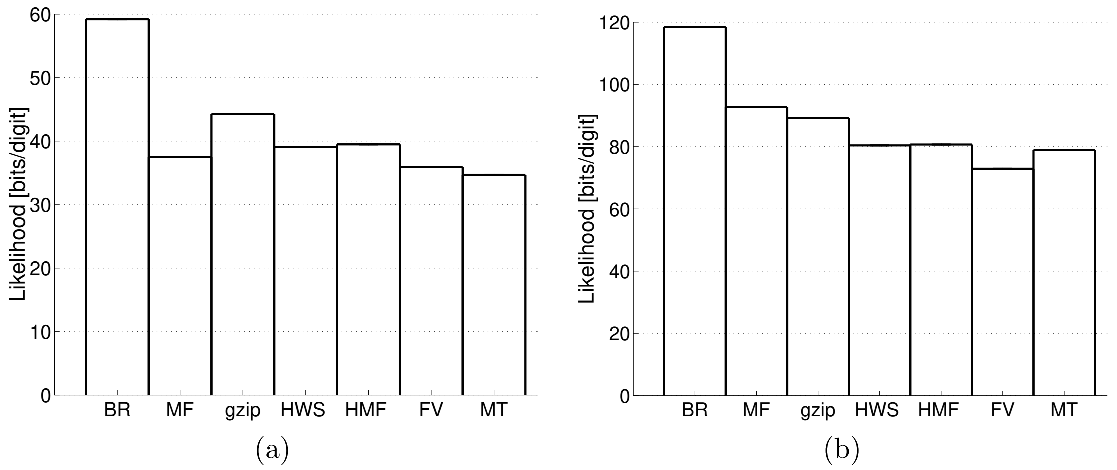


Figure 13: Average log-likelihoods (bits per digit) for the single digit (a) and double digit (b) datasets. Notice the difference in scale between the two figures.

Table 3: Density estimation results for the mixtures of trees and other models on the ALARM data set. Training set size $N_{train} = 10,000$. Average and standard deviation over 20 trials.

Model	Train likelihood [bits/data point]	Test likelihood [bits/data point]
ALARM net	13.148	13.264
Mixture of trees $m = 18$	13.51 ± 0.04	14.55 $\pm$ 0.06
Mixture of factorials $m = 28$	17.11 $\pm$ 0.12	17.64 $\pm$ 0.09
Base rate	30.99	31.17
gzip	40.345	41.260

and "gzip." For MT and MF the model order m and the degree of smoothing were selected by cross validation on randomly selected subsets of the training set.

The results are presented in Table 3, where we see that the MT model outperforms the MF model as well as gzip and the base rate model.

To examine the sensitivity of the algorithms to the size of the data set we ran the same experiment with a training set of size 1,000. The results are presented in Table 4. Again, the MT model is the closest to the true model. Notice that the degradation in performance for the mixture of trees is relatively mild (about 1 bit), whereas the model complexity is reduced significantly. This indicates the important role played by the tree structures in fitting the data and motivates the advantage of the mixture of trees over the mixture of factorials for this data set.

Table 4: Density estimation results for the mixtures of trees and other models on a data set of size 1000 generated from the ALARM network. Average and standard deviation over 20 trials. Recall that α is a smoothing coefficient.

Model	Train likelihood [bits/data point]	Test likelihood [bits/data point]
ALARM net	13.167	13.264
Mixture of trees $m = 2$, $\alpha = 50$	14.56 $\pm$0.16	15.51 $\pm$ 0.11
Mixture of factorials $m = 12$, $\alpha = 100$	18.20 $\pm$ 0.37	19.99 $\pm$ 0.49
Base rate	31.23	31.18
gzip	45.960	46.072

Table 5: Density estimation results for the mixtures of trees and other models on a the FACES data set. Average and standard deviation over 10 trials.

Model	Train likelihood [bits/data point]	Test likelihood [bits/data point]
Mixture of trees $m = 2$, $\alpha = 10$	52.77 $\pm$0.33	56.29 $\pm$ 1.67
Mixture of factorials $m = 24$, $\alpha = 100$	56.34 $\pm$ 0.48	64.41 $\pm$ 2.11
Base rate	75.84	74.27
gzip	–	103.51

5.2.3 The FACES dataset

For the third density estimation experiment, we used a subset of 576 images from a normalized face images dataset (Philips, Moon, Rauss, & Rizvi, 1997). These images were downsampled to 48 variables (pixels) and 5 gray levels. We divided the data randomly into $N_{train} = 500$ and $N_{test} = 76$ examples; of the 500 training examples, 50 were left out as a validation set and used to select m and α for the MT and MF models. The results in table 5 show the mixture of trees as the clear winner. Moreover, the MT achieves this performance with almost 5 times fewer parameters than the second best model, the mixture of 24 factorial distributions.

Note that an essential ingredient of the success of the MT both here and in the digits experiments is that the data are "normalized", i.e., a pixel/variable corresponds approximately to the same location on the underlying digit or face. We do not expect MTs to perform well on randomly chosen image patches.

5.3 Classification with mixtures of trees

5.3.1 Using a mixture of trees as a classifier

A density estimator can be turned into a classifier in two ways, both of them being essentially *likelihood ratio* methods. Denote the class variable by c and the set of input variables by V. In the first method, adopted in our classification experiments under the name of *MT classifier*, an MT model Q is trained on the domain $\{c\} \bigcup V$, treating the class variable like any other variable and pooling all the training data together. In the testing phase, a new instance $x \in \Omega(V)$ is classified by picking the most likely value of the class variable given the settings of the other variables:

$$c(x) \;=\; \operatorname*{argmax}_{x_c} Q(x_c, x)$$

Similarly, for the MF classifier (termed "D-SIDE" by Kontkanen et al., 1996), Q above is an MF trained on $\{c\} \bigcup V$.

The second method calls for partitioning the training set according to the values of the class variable and for training a tree density estimator on each partition. This is equivalent to training a mixture of trees with observed choice variable, the choice variable being the class c (Chow & Liu, 1968; Friedman et al., 1997). In particular, if the trees are forced to have the same structure we obtain the Tree Augmented Naive Bayes (TANB) classifier of Friedman et al. (1997). In either case one turns to Bayes formula:

$$c(x) \;=\; \operatorname*{argmax}_{k} P[c = k]\; T^k(x)$$

to classify a new instance x. The analog of the MF classifier in this setting is the naive Bayes classifier.

5.3.2 The AUSTRALIAN data set

This dataset has 690 examples each consisting of 14 attributes and a binary class variable (Blake & Merz, 1998). In the following we replicated the experimental procedure of Kontkanen et al. (1996) and Michie et al. (1994) as closely as possible. The test and training set sizes were 70 and 620 respectively. For each value of m we ran our algorithm for a fixed number of epochs on the training set and then recorded the performance on the test set. This was repeated 20 times for each m, each time with a random start and with a random split between the test and the training set. Because of the small data set size we used edge pruning with $\beta \propto 1/m$. The best performance of the mixtures of trees is compared to the published results of Kontkanen et al. (1996) and Michie et al. (1994) for the same dataset in Table 6.

5.3.3 The AGARICUS-LEPIOTA dataset

The AGARICUS-LEPIOTA data (Blake & Merz, 1998) comprises 8124 examples, each specifying the 22 discrete attributes of a species of mushroom in the Agaricus and Lepiota families and classifying it as *edible* or *poisonous*. The arities of the variables range from 2 to 12. We created a test set of $N_{test} = 1124$ examples and a training set of $N_{train} = 7000$ examples. Of the latter, 800 examples were kept aside to select m and the rest were used

Table 6: Performance comparison between the MT model and other classification methods on the AUSTRALIAN dataset (Michie et al., 1994). The results for mixtures of factorial distribution are those reported by Kontkanen et al. (1996).

Method	**% correct**	**Method**	**% correct**
Mixture of trees $m = 20$, $\beta = 4$	**87.8**	Backprop	84.6
Mixture of factorial distributions	87.2	C4.5	84.6
Cal5 (decision tree)	86.9	SMART	84.2
ITrule	86.3	Bayes Trees	82.9
Logistic discrimination	85.9	K-nearest neighbor $k = 1$	81.9
Linear discrimination	85.9	AC2	81.9
DIPOL92	85.9	NewID	81.9
Radial basis functions	85.5	LVQ	80.3
CART	85.5	ALLOC80	79.9
CASTLE	85.2	CN2	79.6
Naive Bayes	84.9	Quadratic discrimination	79.3
IndCART	84.8	Flexible Bayes	78.3

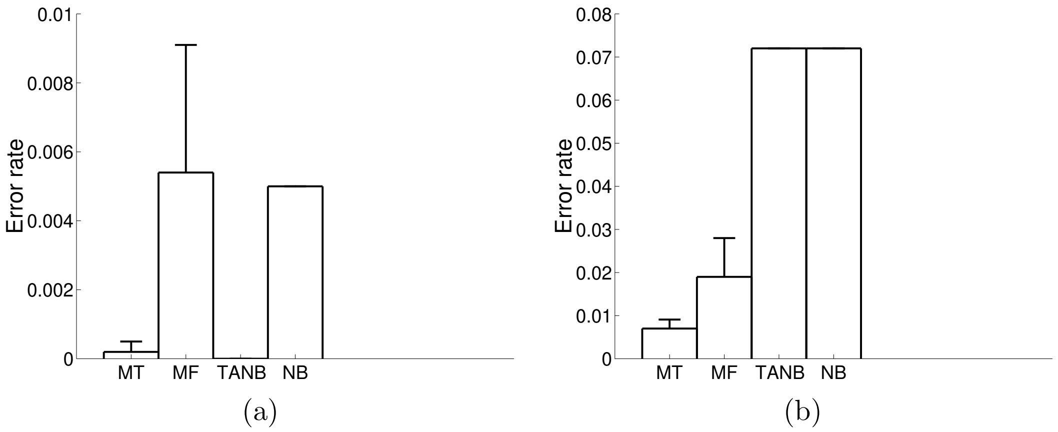


Figure 14: Classification results for the mixtures of trees and other models: (a) On the AGARICUS-LEPIOTA data set; the MT has $m = 12$, and the MF has $m = 30$. (b) On the NURSERY data set; the MT has $m = 30$, the MF has $m = 70$. TANB and NB are the tree augmented naive Bayes and the naive Bayes classifiers respectively. The plots show the average and standard deviation test set error rate over 5 trials.

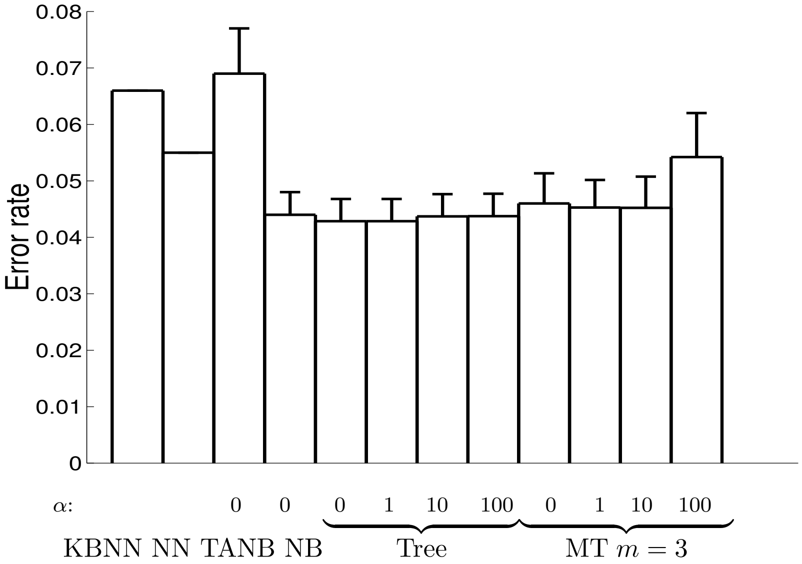


Figure 15: Comparison of classification performance of the MT and other models on the SPLICE data set when $N^{train} = 2000$, $N^{test} = 1175$. Tree represents a mixture of trees with $m = 1$, MT is a mixture of trees with $m = 3$. KBNN is the Knowledge based neural net, NN is a neural net.

for training. No smoothing was used. The classification results on the test set are presented in figure 14(a). As the figure suggests, this is a relatively easy classification problem, where seeing enough examples guarantees perfect performance (achieved by the TANB). The MT (with $m = 12$) achieves nearly optimal performance, making one mistake in one of the 5 trials. The MF and naive Bayes models follow about 0.5% behind.

5.3.4 The NURSERY dataset

This data set contains 12,958 entries (Blake & Merz, 1998), consisting of 8 discrete attributes and one class variable taking 4 values[1]. The data were randomly separated into a training set of size $N_{train} = 11,000$ and a test set of size $N_{test} = 1958$. In the case of MTs and MFs the former data set was further partitioned into 9000 examples used for training the candidate models and 2000 examples used to select the optimal m. The TANB and naive Bayes models were trained on all the 11,000 examples. No smoothing was used since the training set was large. The classification results are shown in figure 14(b).

5.3.5 The SPLICE dataset: Classification

We also studied the classification performance of the MT model in the domain of DNA SPLICE-junctions. The domain consists of 60 variables, representing a sequence of DNA

1. The original data set contains another two data points which correspond to a fifth class value; we eliminated those from the data we used.

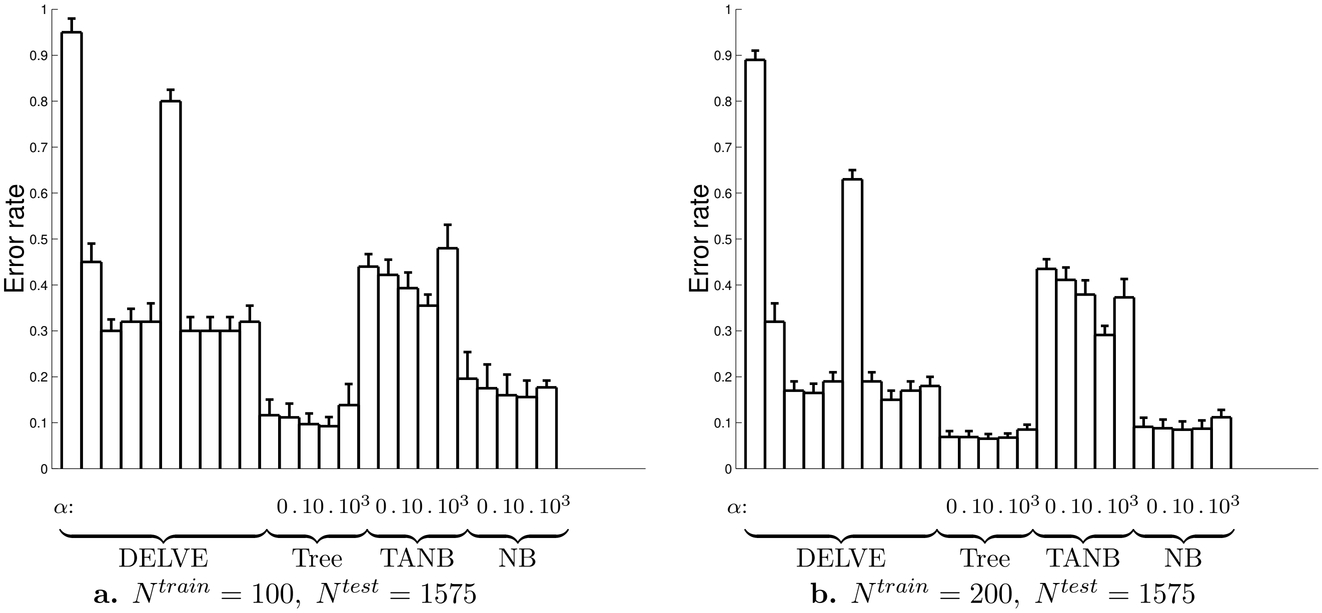


Figure 16: Comparison of classification performance of the mixture of trees and other models trained on small subsets of the SPLICE data set. The models tested by DELVE are, from left to right: 1-nearest neighbor, CART, HME (hierarchical mixture of experts)-ensemble learning, HME-early stopping, HME-grown, K-nearest neighbors, LLS (linear least squares), LLS-ensemble learning, ME (mixture of experts)-ensemble learning, ME-early stopping. TANB is the Tree Augmented Naive Bayes classifier, NB is the Naive Bayes classifier, and Tree is the single tree classifier.

bases, and an additional class variable (Rasmussen et al., 1996). The task is to determine if the middle of the sequence is a splice junction and what is its type. Splice junctions are of two types: exon-intron (EI) represents the end of an exon and the beginning of an intron whereas intron-exon (IE) is the place where the intron ends and the next exon, or coding section, begins. Hence, the class variable can take 3 values (EI, IE or no junction) and the other variables take 4 values corresponding to the 4 possible DNA bases (C, A, G, T). The dataset consists of 3,175 labeled examples.[2]

We ran two series of experiments comparing the MT model with competing models. In the first series of experiments, we compared to the results of Noordewier et al. (1991), who used multilayer neural networks and knowledge-based neural networks for the same task. We replicated these authors' choice of training set size (2000) and test set size (1175) and sampled new training/test sets for each trial. We constructed trees ($m = 1$) and mixtures of trees ($m = 3$). In fitting the mixture, we used an early-stopping procedure in which N_{valid}=300 examples were separated out of the training set and training was stopped when the likelihood on these examples stopped increasing. The results, averaged over 20 trials, are presented in Figure 15 for a variety of values of α. It can be seen that the single tree and the MT model perform similarly, with the single tree showing an insignificantly better classification accuracy. Note that in this situation smoothing does not improve performance; this is not unexpected since the data set is relatively large. With the exception of the "oversmoothed" MT model ($\alpha = 100$), all the single tree or MT models outperform the other models tested on this problem. Note that whereas the tree models contain no prior knowledge about the domain, the other two models do: the neural network model is trained in supervised mode, optimizing for class accuracy, and the KBNN includes detailed domain knowledge.

Based on the strong showing of the single tree model on the SPLICE task, we pursued a second series of experiments in which we compare the tree model with a larger collection of methods from the DELVE repository (Rasmussen et al., 1996). The DELVE benchmark uses subsets of the SPLICE database with 100 and 200 examples for training. Testing is done on 1500 examples in all cases. Figure 16 presents the results for the algorithms tested by DELVE as well as the single trees with different degrees of smoothing. We also show results for naive Bayes (NB) and Tree Augmented Naive Bayes (TANB) models (Friedman et al., 1997). The results from DELVE represent averages over 20 runs with different random initializations on the same training and testing sets; for trees, NB and TANB, whose outputs are not initialization-dependent, we averaged the performance of the models learned for 20 different splits of the union of the training and testing set. No early stopping or cross-validation was used in this case.

The results show that the single tree is quite successful in this domain, yielding an error rate that is less than half of the error rate of the best model tested in DELVE. Moreover, the average error of a single tree trained on 200 examples is 6.9%, which is only 2.3% greater than the average error of the tree trained on 2000 examples. We attempt to explain this striking preservation of accuracy for small training sets in our discussion of feature selection in Section 5.3.7.

2. We eliminated 15 examples from the original data set that had ambiguous inputs (Noordewier, Towell, & Shavlik, 1991; Rasmussen et al., 1996).

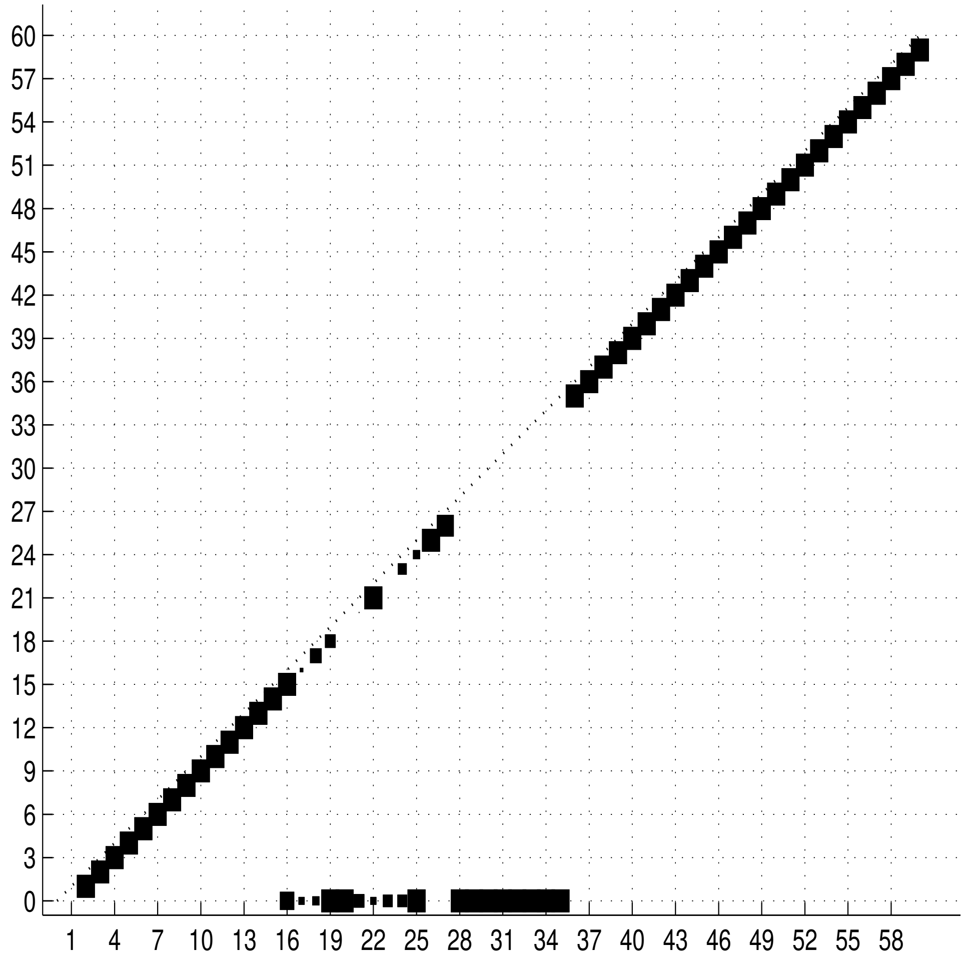


Figure 17: Cumulative adjacency matrix of 20 trees fit to 2000 examples of the SPLICE data set with no smoothing. The size of the square at coordinates ij represents the number of trees (out of 20) that have an edge between variables i and j. No square means that this number is 0. Only the lower half of the matrix is shown. The class is variable 0. The group of squares at the bottom of the figure shows the variables that are connected directly to the class. Only these variable are relevant for classification. Not surprisingly, they are all located in the vicinity of the splice junction (which is between 30 and 31). The subdiagonal "chain" shows that the rest of the variables are connected to their immediate neighbors. Its lower-left end is edge 2–1 and its upper-right is edge 60-59.

EI junction

	Exon			Intron					
	28	29	30	31	32	33	34	35	36
Tree	CA	**A**	**G**	**G**	**T**	AG	A	G	-
True	**CA**	**A**	**G**	**G**	**T**	AG	A	G	T

IE junction

	Intron									Exon
	15	16	...	25	26	27	28	29	30	31
Tree	–	CT	CT	CT	–	–	CT	**A**	**G**	**G**
True	CT	CT	CT	CT	–	–	CT	**A**	**G**	**G**

Figure 18: The encoding of the IE and EI splice junctions as discovered by the tree learning algorithm, compared to the ones given by Watson et al., "Molecular Biology of the Gene" (Watson et al., 1987). Positions in the sequence are consistent with our variable numbering: thus the splice junction is situated between positions 30 and 31. Symbols in boldface indicate bases that are present with probability almost 1, other A,C,G,T symbols indicate bases or groups of bases that have high probability (>0.8), and a – indicates that the position can be occupied by any base with a non-negligible probability.

The Naive Bayes model exhibits behavior that is very similar to the tree model and only slightly less accurate. However, augmenting the Naive Bayes model to a TANB significantly hurts the classification performance.

5.3.6 The SPLICE dataset: Structure identification

Figure 17 presents a summary of the tree structures learned from the $N = 2000$ dataset in the form of a cumulated adjacency matrix. The adjacency matrices of the 20 graph structures obtained in the experiment have been summed. The size of the black square at coordinates i, j in the figure is proportional to the value of the i, j-th element of the cumulated adjacency matrix. No square means that the respective element is 0. Since the adjacency matrix is symmetric, only half of the matrix is shown. From Figure 17 we see that the tree structure is very stable over the 20 trials. Variable 0 represents the class variable; the hypothetical splice junction is situated between variables 30 and 31. The figure shows that the splice junction (variable 0) depends only on DNA sites that are in its vicinity. The sites that are remote from the splice junction are dependent on their immediate neighbors. Moreover, examining the tree parameters, for the edges adjacent to the class variable, we observe that these variables build certain patterns when the splice junction is present, but are random and almost uniformly distributed in the absence of a splice junction. The patterns extracted from the learned trees are shown in Figure 18. The same figure displays the "true" encodings of the IE and EI junctions as given by Watson et al. (1987). The match between the two encodings is almost perfect. Thus, we can conclude that for this

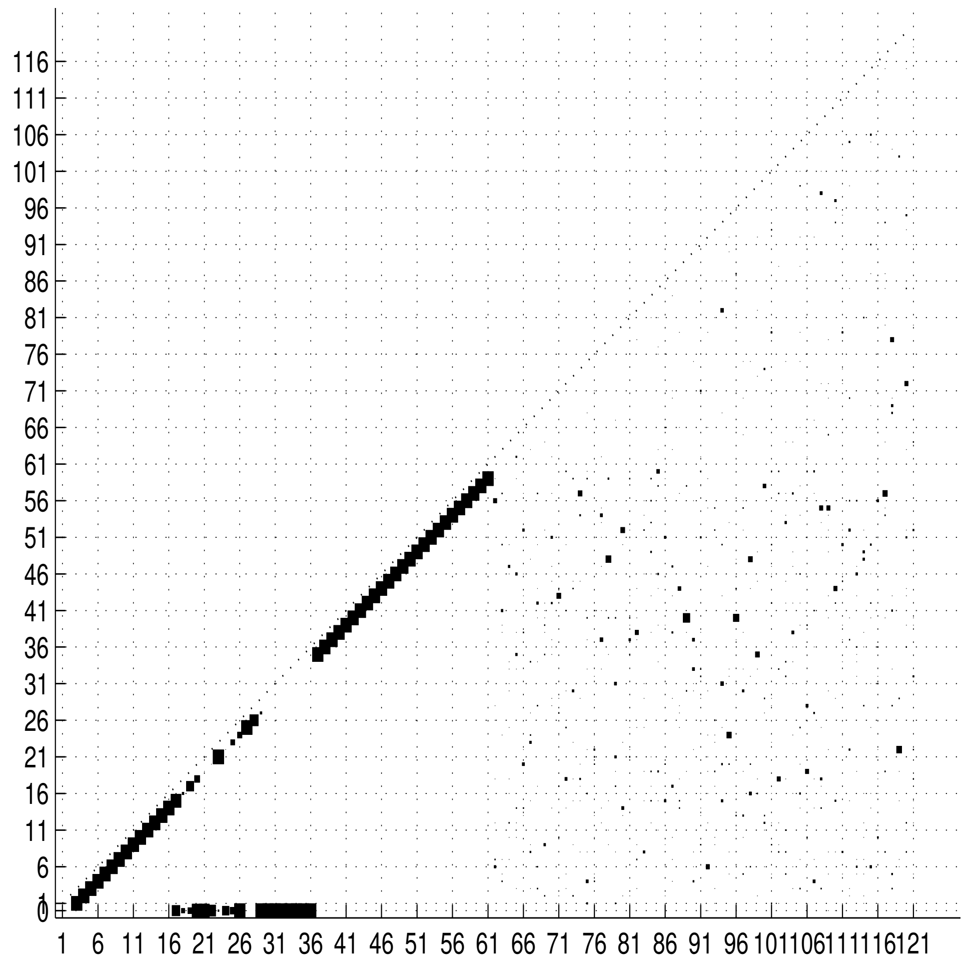


Figure 19: The cumulated adjacency matrix for 20 trees over the original set of variables (0-60) augmented with 60 "noisy" variables (61-120) that are independent of the original ones. The matrix shows that the tree structure over the original variables is preserved.

domain, the tree model not only provides a good classifier but also discovers a model of the physical reality underlying the data. Note that the algorithm arrives at this result in the absence of prior knowledge: (1) it does not know which variable is the class variable, and (2) it does not know that the variables are in a sequence (i.e., the same result would be obtained if the indices of the variables were scrambled).

5.3.7 The SPLICE dataset: Feature selection

Let us examine the single tree classifier that was used for the SPLICE data set more closely. According to the Markov properties of the tree distribution, the probability of the class variables depends only on its neighbors, that is, on the variables to which the class variable is connected by tree edges. Hence, a tree acts as an implicit variable selector for classification: only the variables adjacent to the queried variable (this set of variables is called the *Markov blanket*; Pearl, 1988) are relevant for determining its probability distribution. This property also explains the observed preservation of the accuracy of the tree classifier when the size of the training set decreases: out of the 60 variables, only 18 are relevant to the class; moreover, the dependence is parametrized as 18 independent pairwise probability tables $T_{class,v}$. Such parameters can be fit accurately from relatively few examples. Hence, as long as the training set contains enough data to establish the correct dependency structure, the classification accuracy will degrade slowly with the decrease in the size of the data set.

This explanation helps to understand the superiority of the tree classifier over the models in DELVE: only a small subset of variables are relevant for classification. The tree finds them correctly. A classifier that is not able to perform feature selection reasonably well will be hindered by the remaining irrelevant variables, especially if the training set is small.

For a given Markov blanket, the tree classifies in the same way as a naive Bayes model with the Markov blanket variables as inputs. Note also that the naive Bayes model itself has a built-in feature selector: if one of the input variables v is not relevant to the class, the distributions $P_{v|c}$ will be roughly the same for all values of c. Consequently, in the posterior $P_{c|v}$ that serves for classification, the factors corresponding to v will simplify and thus v will have little influence on the classification. This may explain why the naive Bayes model also performs well on the SPLICE classification task. Notice however that the variable selection mechanisms implemented by the tree classifier and the naive Bayes classifier are not the same.

To verify that indeed the single tree classifier acts like a feature selector, we performed the following experiment, again using the SPLICE data. We augmented the variable set with another 60 variables, each taking 4 values with randomly and independently assigned probabilities. The rest of the experimental conditions (training set, test set and number of random restarts) were identical to the first SPLICE experiment. We fit a set of models with $m = 1$, a small $\beta = 0.1$ and no smoothing. The structure of the new models, in the form of a cumulative adjacency matrix, is shown in Figure 19. We see that the structure over the original 61 variables is unchanged and stable; the 60 noise variables connect in a random uniform patterns to the original variables and among each other. As expected after examining the structure, the classification performance of the new trees is not affected by the newly introduced variables: in fact the average accuracy of the trees over 121 variables is 95.8%, 0.1% higher than the accuracy of the original trees.

6. The accelerated tree learning algorithm

We have argued that the mixture of trees approach has significant advantages over general Bayesian networks in terms of its algorithmic complexity. In particular, the M step of the EM algorithm for mixtures of trees is the ChowLiu algorithm, which scales quadratically in the number of variables n and linearly in the size of the dataset N. Given that the E step is linear in n and N for mixtures of trees, we have a situation in which each pass of the EM algorithm is quadratic in n and linear in N.

Although this time complexity recommends the MT approach for large-scale problems, the quadratic scaling in n becomes problematic for particularly large problems. In this section we propose a method for reducing the time complexity of the MT learning algorithm and demonstrate empirically the large performance gains that we are able to obtain with this method.

As a concrete example of the kind of problems that we have in mind, consider the problem of clustering or classification of documents in information retrieval. Here the variables are words from a vocabulary, and the data points are documents. A document is represented as a binary vector with a component equal to 1 for each word v that is present in the document and equal to 0 for each word v that is not present. In a typical application the number of documents N is of the order of $10^3 - 10^4$, as is the vocabulary size n. Given such numbers, fitting a single tree to the data requires $n^2N \sim 10^9 - 10^{12}$ counting operations.

Note, however, that this domain is characterized by a certain *sparseness*: in particular, each document contains only a relatively small number of words and thus most of the components of its binary vector are 0.

In this section, we show how to take advantage of data sparseness to accelerate the ChowLiu algorithm. We show that in the sparse regime we can often rank order mutual information values without actually computing these values. We also show how to speed up the computation of the sufficient statistics by exploiting sparseness. Combining these two ideas yields an algorithm—the acCL (accelerated Chow and Liu) algorithm—that provides significant performance gains in both running time and memory.

6.1 The acCL algorithm

We first present the acCL algorithm for the case of binary variables, presenting the extension to general discrete variables in Section 6.2. For binary variables we will say that a variable is "on" when it takes value 1; otherwise we say that it is "off". Without loss of generality we assume that a variable is off more times than it is on in the given dataset.

The target distribution P is assumed to be derived from a set of observations of size N. Let us denote by N_v the number of times variable v is on in the dataset and by N_{uv} the number of times variables u and v are simultaneously on. We call each of the latter events a *co-occurrence* of u and v. The marginal P_{uv} of u and v is given by:

$$\begin{aligned}
N \cdot P_{uv}(1,1) &= N_{uv} \\
N \cdot P_{uv}(1,0) &= N_u - N_{uv} \\
N \cdot P_{uv}(0,1) &= N_v - N_{uv} \\
N \cdot P_{uv}(0,0) &= N - N_v - N_u + N_{uv}
\end{aligned}$$

All the information about P that is necessary for fitting the tree is summarized in the counts N, N_v and N_{uv}, $u, v = 1, \ldots, n$, and from now on we will consider P to be represented by these counts. (It is an easy extension to handle non-integer data, such as when the data points are "weighted" by real numbers).

We now define the notion of sparseness that motivates the acCL algorithm. Let us denote by $|x|$ the number of variables that are on in observation x, where $0 \leq |x| \leq n$. Define s, the *data sparseness*

$$s = \max_{i=1,N} |x^i|.$$

If, for example, the data are documents and the variables represent words from a vocabulary, then s represents the maximum number of distinct words in a document. The time and memory requirements of the algorithm that we describe depend on the sparseness s; the lower the sparseness, the more efficient the algorithm. Our algorithm will realize its largest performance gains when $s \ll n, N$.

Recall that the ChowLiu algorithm greedily adds edges to a graph by choosing the edge that currently has the maximal value of mutual information. The algorithm that we describe involves an efficient way to rank order mutual information. There are two key aspects to the algorithm: (1) how to compare mutual information between non-co-occurring variables, and (2) how to compute co-occurrences in a list representation.

6.1.1 Comparing mutual information between non-co-occurring variables

Let us focus on the pairs u, v that do not co-occur, i.e., for which $N_{uv} = 0$. For such a pair, the mutual information I_{uv} is a function of N_u, N_v and N. Let us analyze the variation of the mutual information with respect to N_v by taking the corresponding partial derivative:

$$\frac{\partial I_{uv}}{\partial N_v} \;=\; \log \frac{N - N_v}{N - N_u - N_v} \;>\; 0 \tag{11}$$

This result implies that for a given variable u and any two variables v, v' for which $N_{uv} = N_{uv'} = 0$ we have:

$$N_v > N_{v'} \text{ implies that } I_{uv} > I_{uv'}.$$

This observation allows us to partially sort the mutual information values I_{uv} for non-co-occurring pairs u, v, without computing them. First, we have to sort all the variables by their number of occurrences N_v. We store the result in a list L. This gives a total ordering "$\succ$" for the variables in V:

$$v \succ u \;\Leftrightarrow\; v \text{ preceeds } u \text{ in list } L \;\Rightarrow\; N_v \;\geq\; N_u.$$

For each u, we define the list of variables following u in the ordering "$\succ$" and not co-occurring with it:

$$V_0(u) \;=\; \{v \in V,\; v \prec u,\; N_{uv} = 0\}.$$

This list is sorted by decreasing N_v and therefore, implicitly, by decreasing I_{uv}. Since the data are sparse, most pairs of variables do not co-occur. Therefore, by creating the lists $V_0(u)$, a large number of values of the mutual information are partially sorted. Before showing how to use this construction, let us examine an efficient way of computing the N_{uv} counts when the data are sparse.

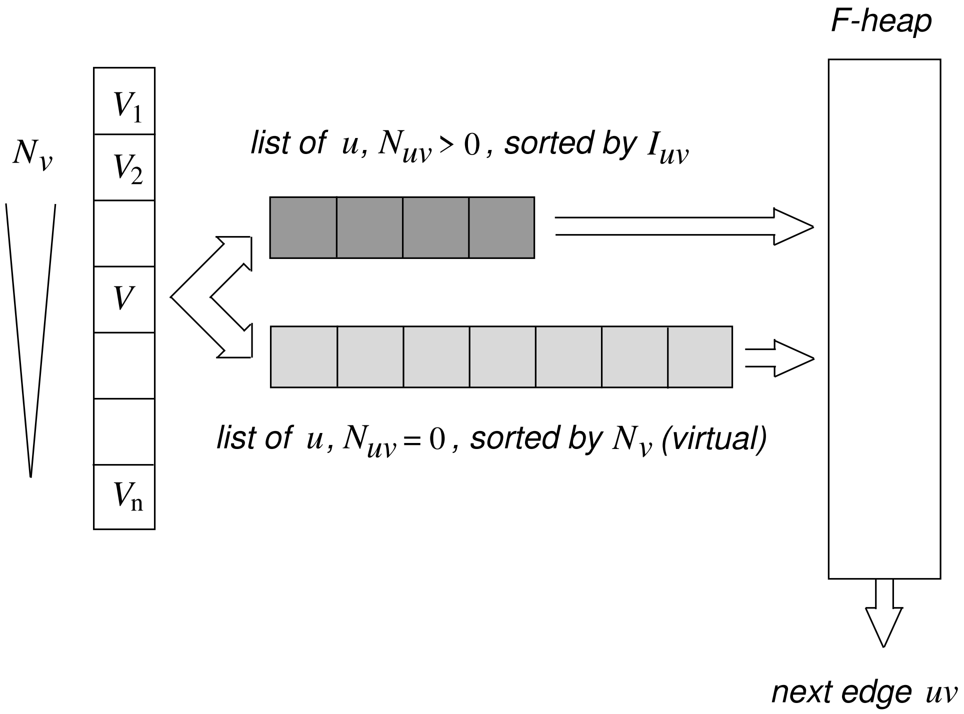


Figure 20: The data structure that supplies the next candidate edge. Vertically on the left are the variables, sorted by decreasing N_u. For a given u, there are two lists: $C(u)$, sorted by decreasing I_{uv} and (the virtual list) $V_0(u)$, sorted by decreasing N_v. The maximum of the two first elements of these lists is inserted into an Fibonacci heap. The overall maximum of I_{uv} can then be extracted as the maximum of the Fibonacci heap.

6.1.2 Computing co-occurrences in a list representation

Let $\mathcal{D} = \{x^1, \ldots x^N\}$ be a set of observations over n binary variables. If $s \ll n$ it is efficient to represent each observation in $\mathcal{D}$ as a list of the variables that are on in the respective observation. Thus, data point $x^i, i = 1, \ldots N$ will be represented by the list $xlist^i = \text{list}\{v \in V | x_v^i = 1\}$. The space required by the lists is no more than sN and is much smaller than the space required by the binary vector representation of the same data (i.e., nN).

Note, moreover, that the total number N_C of co-occurrences in the dataset $\mathcal{D}$ is

$$N_C = \sum_{v \succ u} N_{uv} \leq \frac{1}{2} s^2 N.$$

For the variables that co-occur with u, a set of *co-occurrence lists* $C(u)$ is created. Each $C(u)$ contains records (v, I_{uv}), $v \prec u, N_{uv} > 0$ and is sorted by decreasing I_{uv}. To represent the lists $V_0(u)$ (that contain many elements) we use their "complements" $\overline{V_0}(u) = \{v \in V,\ v \prec u,\ N_{uv} > 0\}$. It can be shown (Meilă-Predoviciu, 1999) that the

Algorithm acCL($\mathcal{D}$)

Input: Variable set V of size n
Dataset $\mathcal{D} = \{xlist^i,\ i = 1, \ldots N\}$
Procedure Kruskal

1. compute N_v for $v \in V$; create L, list of variables in V sorted by decreasing N_v
2. compute co-occurrences N_{uv}; create lists $C(u)$, $\overline{V}_0(u),\ u \in V$
3. create *vheap*
   for $u \in vlist$
   $v = \underset{\text{head}C_u, \text{head}\overline{V}_0(u)}{\text{argmax}} I_{uv}$
   insert (N_{uv}, u, v, I_{uv}) in *vheap*
4. E = Kruskal(*vheap*); store N_{uv} for the edges uv added to E
5. for $(u, v) \in E$ compute the probability table T_{uv} using N_u, N_v, N_{uv} and N.

Output: T

Figure 21: The acCL algorithm.

computation of the co-occurrence counts and the construction of the lists $C(u)$ and $\overline{V}_0(u)$, for all $u \in V$, takes an amount of time proportional to the number of co-occurrences N_C, up to a logarithmic factor:

$$\mathcal{O}(s^2 N \log(s^2 N/n)).$$

Comparing this value with $\mathcal{O}(n^2 N)$, which is the time to compute P_{uv} in the ChowLiu algorithm, we see that the present method replaces the dimension of the domain n by s. The memory requirements for the lists are also at most proportional to N_C, hence $\mathcal{O}(s^2 N)$.

6.1.3 The algorithm and its data structures.

We have described efficient methods for computing the co-occurrences and for partially sorting the mutual information values. What we aim to create is a mechanism that will output the edges (u, v) in decreasing order of their mutual information.

We shall set up this mechanism in the form of a *Fibonacci heap* (Fredman & Tarjan, 1987) called *vheap* that contains an element for each $u \in V$, represented by the edge with the highest mutual information among the edges (u, v), with $v \prec u$, that are not yet eliminated. The record in *vheap* is of the form (N_{uv}, u, v, I_{uv}), with $v \prec u$, and with I_{uv} being the key used for sorting. Once the maximum is extracted, the used edge has to be replaced by the next largest (in terms of I_{uv}) edge in u's lists.

To perform this latter task we use the data structures shown in Figure 20. Kruskal's algorithm is now used to construct the desired spanning tree. Figure 21 summarizes the resulting algorithm.

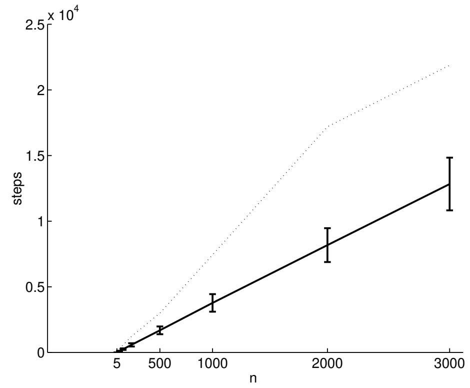


Figure 22: The mean (full line), standard deviation and maximum (dotted line) of the number of steps n_K in the Kruskal algorithm over 1000 runs plotted against $n \log n$. n ranges from 5 to 3000. The edge weights were sampled from a uniform distribution.

6.1.4 Running time

The algorithm requires: $\mathcal{O}(sn)$ for computing $N_v, v \in V$, $\mathcal{O}(n \log n)$ for sorting the variables, $\mathcal{O}(s^2 N \log(s^2 N/n))$ for step 2, $\mathcal{O}(n)$ for step 3, $\mathcal{O}(n_K \log n + N_C)$ for the Kruskal algorithm (n_K is the number of edges examined by the algorithm, $\log n$ is the time for an extraction from *vheap*, N_C is an upper bound for the number of elements in the lists $\overline{V}_0$ whose elements need to be skipped occasionally when we extract variables from the virtual lists V_0), and $\mathcal{O}(n)$ for creating the the $n-1$ probability tables in step 5. Summing these terms, we obtain the following upper bound for the running time of the acCL algorithm:

$$\mathcal{O}(n \log n + sn + s^2 N \log \frac{s^2 N}{n} + n_K \log n).$$

If we ignore the logarithmic factors this simplifies to

$$\tilde{\mathcal{O}}(sn + s^2 N + n_K).$$

For constant s, this bound is a polynomial of degree 1 in the three variables n, N and n_K. Because n_K has the range $n - 1 \leq n_K \leq n(n-1)/2$, the worst case complexity of the AcCL algorithm is quadratic in n. Empirically, however, as we show below, we find that the dependence of n_K on n is generally subquadratic. Moreover, random graph theory implies that if the distribution of the weight values is the same for all edges, then Kruskal's algorithm should take time proportional to $n \log n$ (West, 1996). To verify this latter result, we conducted a set of Monte Carlo experiments, in which we ran the Kruskal algorithm on sets of random weights over domains of dimension up to $n = 3000$. For each n, 1000 runs were performed. Figure 22 plots the average and maximum n_K versus $n \log n$ for these experiments. The curve for the average displays an essentially linear dependence.

6.1.5 Memory requirements

Beyond the storage for data and results, we need $\mathcal{O}(N_C)$ space to store the co-occurrences and the lists $C(u), \overline{V}_0(u)$ and $\mathcal{O}(n)$ for $L, vheap$ and the Kruskal algorithm. Hence, the additional space used by the acCL algorithm is $\mathcal{O}(s^2N + n)$.

6.2 Discrete variables of arbitrary arity

We briefly describe the extension of the acCL algorithm to the case of discrete domains in which the variables can take more than two values.

First we extend the definition of data sparseness: we assume that for each variable there exists a special value that appears with higher frequency than all the other values. This value will be denoted by 0, without loss of generality. For example, in a medical domain, the value 0 for a variable would represent the "normal" value, whereas the abnormal values of each variable would be designated by non-zero values. An "occurrence" for variable v will be the event $v \neq 0$ and a "co-occurrence" of u and v means that u and v are both non-zero for the same data point. We define $|x|$ as the number of non-zero values in observation x. The sparseness s is, as before, the maximum of $|x|$ over the data set.

To exploit the high frequency of the zero values we represent only the occurrences explicitly, creating thereby a compact and efficient data structure. We obtain performance gains by presorting mutual information values for non-co-occurring variables.

6.2.1 Computing co-occurrences

As before, we avoid representing zero values explicitly by replacing each data point x by the list $xlist$, where $xlist \;=\; \text{list}\{(v, x_v), v \in V, x_v \neq 0\}$.

A co-occurrence is represented by the quadruple (u, x_u, v, x_v), $x_u, x_v \neq 0$. Instead of one co-occurrence count N_{uv}, we now have a two-way contingency table N_{uv}^{ij}. Each N_{uv}^{ij} represents the number of data points where $u = i, v = j$, $i, j \neq 0$. Counting and storing co-occurrences can be done in the same time as before and with a $\mathcal{O}(r_{MAX}^2)$ larger amount of memory, necessitated by the additional need to store the (non-zero) variable values.

6.2.2 Presorting mutual information values

Our goal is to presort the mutual information values I_{uv} for all $v \prec u$ that do not co-occur with u. The following theorem shows that this can be done exactly as before.

Theorem *Let u, v, w be discrete variables such that v, w do not co-occur with u (i.e. $u \neq 0 \;\Rightarrow\; v = w = 0$) in a given dataset $\mathcal{D}$. Let N_{v0}, N_{w0} be the number of datapoints for which $v = 0$ and $w = 0$ respectively, and let I_{uv}, I_{uw} be the respective empirical mutual information values based on the sample $\mathcal{D}$. Then*

$$N_{v0} \;>\; N_{w0} \;\;\Rightarrow\;\; I_{uv} \;\leq\; I_{uw}$$

with equality only if u is identically 0. ■

The proof of the theorem is given in the Appendix. The implication of this theorem is that the acCL algorithm can be extended to variables taking more than two values by making only one (minor) modification: the replacement of the scalar counts N_v and N_{uv} by the vectors N_v^j, $j \neq 0$ and, respectively, the contingency tables N_{uv}^{ij}, $i, j \neq 0$.

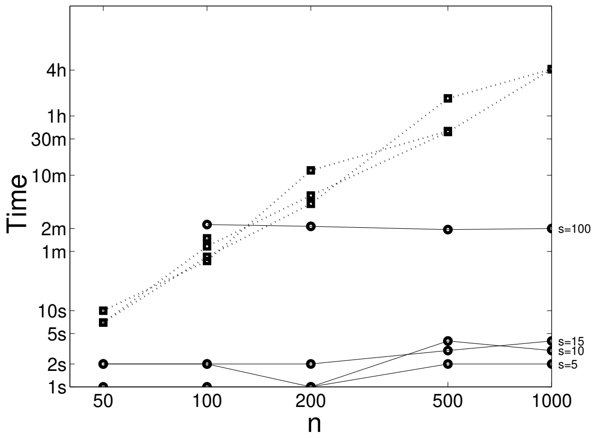


Figure 23: Running time for the acCL (full line) and (dotted line) ChowLiu algorithms versus number of vertices n for different values of the sparseness s.

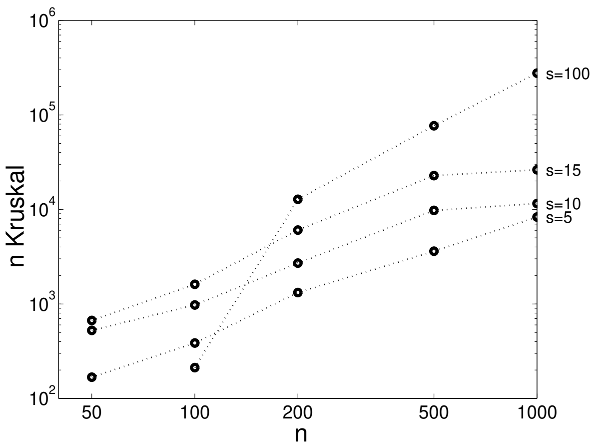


Figure 24: Number of steps of the Kruskal algorithm n_K versus domain size n measured for the acCL algorithm for different values of s.

6.3 Experiments

In this section we report on the results of experiments that compare the speed of the acCL algorithm with the standard ChowLiu method on artificial data.

In our experiments the binary domain dimension n varies from 50 to 1000. Each data point has a fixed number s of variables taking value 1. The sparseness s takes the values 5, 10, 15 and 100. The data were generated from an artificially constructed non-uniform, non-factored distribution. For each pair (n, s) a set of 10,000 points was created.

For each data set both the ChowLiu algorithm and the acCL algorithm were used to fit a single tree distribution. The running times are plotted in Figure 23. The improvements of acCL over the standard version are spectacular: learning a tree on 1000 variables from 10,000 data points takes 4 hours with the standard algorithm and only 4 seconds with the accelerated version when the data are sparse ($s \leq 15$). For the less sparse regime of $s = 100$ the acCL algorithm takes 2 minutes to complete, improving on the traditional algorithm by a factor of 120.

Note also that the running time of the accelerated algorithm seems to be nearly independent of the dimension of the domain. Recall on the other hand that the number of steps n_K (Figure 24) grows with n. This implies that the bulk of the computation lies in the steps preceding the Kruskal algorithm proper. Namely, it is in the computing of co-occurrences and organizing the data that most of the time is spent. Figure 23 also confirms that the running time of the traditional ChowLiu algorithm grows quadratically with n and is independent of s.

This concludes the presentation of the acCL algorithm. The method achieves its performance gains by exploiting characteristics of the data (sparseness) and of the problem (the weights represent mutual information) that are external to the maximum weight spanning tree algorithm proper. The algorithm obtained is worst case $\tilde{\mathcal{O}}(sn+s^2N+n^2)$ but typically $\tilde{\mathcal{O}}(sn+s^2N)$, which represents a significant asymptotic improvement over the $\mathcal{O}(n^2N)$ of the traditional Chow and Liu algorithm. Moreover, if s is large, then the acCL algorithm (gracefully) degrades to the standard ChowLiu algorithm.

The algorithm extends to non-integer counts, hence being directly applicable to mixtures of trees. As we have seen empirically, a very significant part of the running time is spent in computing co-occurrences. This prompts future work on learning statistical models over large domains that focus on the efficient computation and usage of the relevant sufficient statistics. Work in this direction includes the structural EM algorithm (Friedman, 1998; Friedman & Getoor, 1999) as well as A-D trees (Moore & Lee, 1998). The latter are closely related to our representation of the pairwise marginals P_{uv} by counts. In fact, our representation can be viewed as a "reduced" A-D tree that stores only pairwise statistics. Consequently, when an A-D tree representation is already computed, it can be exploited in steps 1 and 2 of the acCL algorithm. Other versions of the acCL algorithm are discussed by Meilă-Predoviciu (1999).

7. Conclusions

We have presented the mixture of trees (MT), a probabilistic model in which joint probability distributions are represented as finite mixtures of tree distributions. Tree distributions have a number of virtues—representational, computational and statistical—but have limited expressive power. Bayesian and Markov networks achieve significantly greater expressive power while retaining many of the representational virtues of trees, but incur significantly higher costs on the computational and statistical fronts. The mixture approach provides an alternative upgrade path. While Bayesian and Markov networks have no distinguished relationships between edges, and statistical model selection procedures for these networks generally involve additions and deletions of single edges, the MT model groups overlapping sets of edges into mixture components, and edges are added and removed via a maximum likelihood algorithm that is constrained to fit tree models in each mixture component. We have also seen that it is straightforward to develop Bayesian methods that allow finer control over the choice of edges and smooth the numerical parameterization of each of the component models.

Chow and Liu (1968) presented the basic maximum likelihood algorithm for fitting tree distributions that provides the M step of our EM algorithm, and also showed how to use ensembles of trees to solve classification problems, where each tree models the class-conditional density of one of the classes. This approach was pursued by Friedman et al. (1997, 1998), who emphasized the connection with the naive Bayes model, and presented empirical results that demonstrated the performance gains that could be obtained by enhancing naive Bayes to allow connectivity between the attributes. Our work is a further contribution to this general line of research—we treat an ensemble of trees as a mixture distribution. The mixture approach provides additional flexibility in the classification domain, where the "choice variable" need not be the class label, and also allows the architecture to be applied to unsupervised learning problems.

The algorithms for learning and inference that we have presented have relatively benign scaling: inference is linear in the dimensionality n and each step of the EM learning algorithm is quadratic in n. Such favorable time complexity is an important virtue of our tree-based approach. In particularly large problems, however, such as those that arise in information retrieval applications, quadratic complexity can become onerous. To allow the use of the MT model to such cases, we have developed the acCL algorithm, where by exploiting data sparseness and paying attention to data structure issues we have significantly reduced run time: We presented examples in which the speed-up obtained from the acCL algorithm was three orders of magnitude.

Are there other classes of graphical models whose structure can be learned efficiently from data? Consider the class of Bayesian networks for which the topological ordering of the variables is fixed and the number of parents of each node is bounded by a fixed constant l. For this class the optimal model structure for a given target distribution can be found in $\mathcal{O}(n^{l+1})$ time by a greedy algorithm. These models share with trees the property of being *matroids* (West, 1996). The matroid is the unique algebraic structure for which the "maximum weight" problem, in particular the maximum weight spanning tree problem, is solved optimally by a greedy algorithm. Graphical models that are matroids have efficient structure learning algorithms; it is an interesting open problem to find additional examples of such models.

Acknowledgments

We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-9988642) and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637).

Appendix A.

In this appendix we prove the following theorem from Section 6.2:
Theorem *Let u, v, w be discrete variables such that v, w do not co-occur with u (i.e., $u \neq 0 \;\Rightarrow\; v = w = 0$ in a given dataset $\mathcal{D}$). Let N_{v0}, N_{w0} be the number of data points for which $v = 0, w = 0$ respectively, and let I_{uv}, I_{uw} be the respective empirical mutual information values based on the sample $\mathcal{D}$. Then*

$$N_{v0} \;>\; N_{w0} \;\;\Rightarrow\;\; I_{uv} \;\leq\; I_{uw}$$

with equality only if u is identically 0. ■
Proof. We use the notation:

$$P_v(i) \;=\; \frac{N_v^i}{N}, \quad i \neq 0; \quad P_{v0} \;\equiv\; P_v(0) \;=\; 1 - \sum_{i \neq 0} P_v(i).$$

These values represent the (empirical) probabilities of v taking value $i \neq 0$ and 0 respectively. Entropies will be denoted by H. We aim to show that $\frac{\partial I_{uv}}{\partial P_{v0}} < 0$.

We first note a "chain rule" expression for the entropy of a discrete variable. In particular, the entropy H_v of any multivalued discrete variable v can be decomposed in the following way:

$$\begin{aligned} H_v &= \underbrace{-P_{v0}\log P_{v0} - (1-P_{v0})\log(1-P_{v0})}_{H_{v0}} \\ &\quad \underbrace{-(1-P_{v0})\sum_{i\neq 0}\frac{P_v(i)}{(1-P_{v0})}\log\frac{P_v(i)}{(1-P_{v0})}}_{-H_{\overline{v}}} \\ &= H_{v0} + (1-P_{v0})H_{\overline{v}}. \end{aligned} \tag{12}$$

Note moreover that the mutual information of two non-co-occurring variables is $I_{uv} = H_u - H_{u|v}$. The second term, the conditional entropy of u given v is

$$H_{u|v} \;=\; P_{v0}H_{u|v=0} + \sum_{j\neq 0} P_v(j)\,\underbrace{H_{u|v=j}}_{0}.$$

We now expand $H_{u|v=0}$ using the decomposition in (12):

$$H_{u|v=0} \;=\; H_{u0|v=0} + (1 - P_{u=0|v=0})H_{u|u\neq 0, v=0}.$$

Because u and v are never non-zero at the same time, all non-zero values of u are paired with zero values of v. Consequently $Pr[u = i|u \neq 0, v = 0] = Pr[u = i|u \neq 0]$ and $H_{u|u\neq 0,v=0} = H_{\overline{u}}$. The term denoted $H_{u0|v=0}$ is the entropy of a binary variable whose probability is $Pr[u = 0|v = 0]$. This probability equals

$$Pr[u = 0|v = 0] \;=\; 1 - \frac{1 - P_{u0}}{1 - P_{v0}}.$$

Note that in order to obtain a non-negative probability in the above equation one needs $1 - P_{u0} \leq P_{v0}$, a condition that is always satisfied if u and v do not co-occur. Replacing the previous three equations in the formula of the mutual information, we get

$$I_{uv} = P_{u0} \log P_{u0} - P_{v0} \log P_{v0} + (P_{u0} + P_{v0} - 1) \log(P_{u0} + P_{v0} - 1).$$

This expression, remarkably, depends only on P_{u0} and P_{v0}. Taking its partial derivative with respect to P_{v0} yields

$$\frac{\partial I_{uv}}{\partial P_{v0}} = \log \frac{P_{v0} + P_{u0} - 1}{P_{v0}} \leq 0,$$

a value that is always negative, independently of P_{v0}. This shows the mutual information increases monotonically with the "occurrence frequency" of v given by $1 - P_{v0}$. Note also that the above expression for the derivative is the same as the result obtained for binary variables in (11).

References

Bishop, C. M. (1999). Latent variable models. In M. I. Jordan (Ed.), *Learning in Graphical Models.* Cambridge, MA: MIT Press.

Blake, C., & Merz, C. (1998). *UCI Repository of Machine Learning Databases.* http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in AI* (pp. 64–72). Morgan Kaufmann.

Buntine, W. (1996). A guide to the literature on learning graphical models. *IEEE Transactions on Knowledge and Data Engineering*, *8*, 195–210.

Cheeseman, P., & Stutz, J. (1995). Bayesian classification (AutoClass): Theory and results. In U. Fayyad, G. Piatesky-Shapiro, P. Smyth, & Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 153–180). AAAI Press.

Cheng, J., Bell, D. A., & Liu, W. (1997). Learning belief networks from data: an information theory based approach. In *Proceedings of the Sixth ACM International Conference on Information and Knowledge Management.*

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *IT-14*(3), 462–467.

Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Cormen, T. H., Leiserson, C. E., & Rivest, R. R. (1990). *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., & Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems.* New York, NY: Springer.

Dayan, P., & Zemel, R. S. (1995). Competition and multiple cause models. *Neural Computation*, *7*(3), 565–579.

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

Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, *34*(3), 596–615.

Frey, B. J., Hinton, G. E., & Dayan, P. (1996). Does the wake-sleep algorithm produce good density estimators? In D. Touretzky, M. Mozer, & M. Hasselmo (Eds.), *Neural Information Processing Systems* (pp. 661–667). Cambridge, MA: MIT Press.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proceedings of the 14th Conference on Uncertainty in AI* (pp. 129–138). San Francisco, CA: Morgan Kaufmann.

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

Friedman, N., & Getoor, L. (1999). Efficient learning using constrained sufficient statistics. In *Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics (AISTATS-99).*

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1996). Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1300–1307).

Friedman, N., Goldszmidt, M., & Lee, T. (1998). Bayesian network classification with continous attributes: Getting the best of both discretization and parametric fitting. In *Proceedings of the International Conference on Machine Learning (ICML).*

Geiger, D. (1992). An entropy-based learning algorithm of Bayesian conditional trees. In *Proceedings of the 8th Conference on Uncertainty in AI* (pp. 92–97). Morgan Kaufmann Publishers.

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

Hastie, T., & Tibshirani, R. (1996). Discriminant analysis by mixture modeling. *Journal of the Royal Statistical Society B*, *58*, 155–176.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: the combination of knowledge and statistical data. *Machine Learning*, *20*(3), 197–243.

Hinton, G. E., Dayan, P., Frey, B., & Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, *268*, 1158–1161.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition.* Cambridge, MA: MIT Press.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kontkanen, P., Myllymaki, P., & Tirri, H. (1996). *Constructing Bayesian finite mixture models by the EM algorithm* (Tech. Rep. No. C-1996-9). University of Helsinki, Department of Computer Science.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Lauritzen, S. L. (1996). *Graphical Models.* Oxford: Clarendon Press.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., & Leimer, H.-G. (1990). Independence properties of directed Markov fields. *Networks*, *20*, 579–605.

MacLachlan, G. J., & Bashford, K. E. (1988). *Mixture Models: Inference and Applications to Clustering.* NY: Marcel Dekker.

Meilă, M., & Jaakkola, T. (2000). Tractable Bayesian learning of tree distributions. In C. Boutilier & M. Goldszmidt (Eds.), *Proceedings of the 16th Conference on Uncertainty in AI* (pp. 380–388). San Francisco, CA: Morgan Kaufmann.

Meilă, M., & Jordan, M. I. (1998). Estimating dependency structure as a hidden variable. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Neural Information Processing Systems* (pp. 584–590). MIT Press.

Meilă-Predoviciu, M. (1999). *Learning with mixtures of trees.* Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification.* New York: Ellis Horwood.

Monti, S., & Cooper, G. F. (1998). *A Bayesian network classfier that combines a finite mixture model and a naive Bayes model* (Tech. Rep. No. ISSP-98-01). University of Pittsburgh.

Moore, A. W., & Lee, M. S. (1998). Cached sufficient statistics for efficient machine learning with large datasets. *Journal for Artificial Intelligence Research*, *8*, 67–91.

Neal, R. M., & Hinton, G. E. (1999). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in Graphical Models* (pp. 355–368). Cambridge, MA: MIT Press.

Ney, H., Essen, U., & Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, *8*, 1–38.

Noordewier, M. O., Towell, G. G., & Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In R. P. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems* (pp. 530–538). Morgan Kaufmann Publishers.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufman Publishers.

Philips, P., Moon, H., Rauss, P., & Rizvi, S. (1997). The FERET evaluation methodology for face-recognition algorithms. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition.* San Juan, Puerto Rico.

Rasmussen, C. E., Neal, R. M., Hinton, G. E., Camp, D. van, Revow, M., Ghahramani, Z., Kustra, R., & Tibshrani, R. (1996). *The DELVE Manual.* http://www.cs.utoronto.ca/~delve.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry.* New Jersey: World Scientific Publishing Company.

Rubin, D. B., & Thayer, D. T. (1983). EM algorithms for ML factor analysis. *Psychometrika*, *47*, 69–76.

Saul, L. K., & Jordan, M. I. (1999). A mean field learning algorithm for unsupervised neural networks. In M. I. Jordan (Ed.), *Learning in Graphical Models* (pp. 541–554). Cambridge, MA: MIT Press.

Shafer, G., & Shenoy, P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, *2*, 327–352.

Smyth, P., Heckerman, D., & Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, *9*, 227–270.

Thiesson, B., Meek, C., Chickering, D. M., & Heckerman, D. (1997). *Learning mixtures of Bayes networks* (Tech. Rep. Nos. MSR–POR–97–30). Microsoft Research.

Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., & Weiner, A. M. (1987). *Molecular Biology of the Gene* (Vol. I, 4 ed.). Menlo Park, CA: The Benjamin/Cummings Publishing Company.

West, D. B. (1996). *Introduction to Graph Theory.* Upper Saddle River, NJ: Prentice Hall.