MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1648 September, 1998
C.B.C.L. Memo No. 165

Estimating Dependency Structure as a Hidden Variable

Marina Meilă **Michael I. Jordan** **Quaid Morris**

Abstract

This paper introduces a probability model, the *mixture of trees* that can account for sparse, dynamically changing dependence relationships. We present a family of efficient algorithms based on the EM and the Minimum Spanning Tree algorithms that learn mixtures of trees in the ML framework. The method can be extended to take into account priors and, for a wide class of priors that includes the Dirichlet and the MDL priors, it preserves its computational efficiency. Experimental results demonstrate the excellent performance of the new model both in density estimation and in classification. Finally, we show that a single tree classifier acts like an implicit feature selector, thus making the classification performance insensitive to irrelevant attributes.

This report describes research done at the Dept. of Electrical Engineering and Computer Science, the Dept. of Brain and Cognitive Sciences, the Center for Biological and Computational Learning and the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Dept. of Defense and by the Office of Naval Research. Michael I. Jordan is a NSF Presidential Young Investigator. The authors can be reached at M.I.T., Center for Biological and Computational Learning, 45 Carleton St., Cambridge MA 02142, USA. E-mail: mmp@ai.mit.edu, jordan@psyche.mit.edu, quaid@ai.mit.edu.

1 INTRODUCTION

A fundamental feature of a good model is the ability to uncover and exploit independencies in the data it is presented with. For many commonly used models, such as neural nets and belief networks, the dependency structure encoded in the model is fixed, in the sense that it is not allowed to vary depending on actual values of the variables or with the current case. However, dependency structures that are conditional on values of variables abound in the world around us. Consider for example bitmaps of handwritten digits. They 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.

Models that are able to represent data conditioned dependencies are decision trees and mixture models, including the soft counterpart of the decision tree, the mixture of experts. Decision trees however can only represent certain patterns of dependecy, and in particular are not suited for representing sparse dependencies. Mixtures are more flexible and the rest of this paper will be focusing on one special case called *the mixture of spanning trees*.

We will consider domains where the observed variables are related by pairwise dependencies only and these dependencies are sparse enough to contain no cycles. Therefore they can be represented graphically as a *tree*. The structure of the dependencies may vary from one instance to the next. We index the set of possible dependency structures by a *structure variable* z (that can be observed or hidden) thereby obtaining a *mixture*.

In the framework of graphical probability models, tree distributions enjoy many properties that make them attractive as modelling tools: they have a flexible topology, are intuitively appealing, sampling and computing likelihoods are linear time, simple efficient algorithms for marginalizing and conditioning (quadratic or less in the dimension of the problem) exist. Fitting the best tree to a given distribution can be done exactly and efficiently. Trees can capture simple pairwise interactions between variables but they can prove insufficient for more complex distributions. Mixtures of trees enjoy most of the computational advantages of trees and, in addition, they are universal approximators over the space of all distributions. Therefore, they are appropriate as density estimators for domains where the dependency patterns become tree like when a possibly hidden variable is instantiated. But given the rich history of mixture models as classifiers they appear promising for these task as well.

Mixture models have been extensively used in the statistics and neural network literature. Of relevance to the present work are the mixtures of Gaussians, whose distribution space, in the case of continuous variables overlaps with the space of mixtures of trees. Mixtures of factorial distributions, a subclass of tree distributions, have been investigated recently by [9]. Work on fitting a tree to a distribution in a Maximum-Likelihood (ML) framework has been pioneered by Chow and Liu [2] and was extended to polytrees by Pearl [13] and to mixtures of trees with observed structure variable by Geiger [6] and Friedman [5].

This work presents efficient algorithms for learning mixture of trees models with unknown or hidden structure variable. The following section introduces the model; then, section 3 develops the basic algorithm for its estimation from data in the ML framework. Section 4 discusses the introduction of priors over mixtures of trees models and presents several realistic factorized and non-factorized priors for which the MAP estimate can be computed by modified versions of the basic algorithm. The properties of the model as a density estimator are verified by experiments in section 5 while section 6 studies its classification performance. Section 7 concludes the paper.

2 THE MIXTURE OF TREES MODEL

In this section we will introduce the mixture of trees model and the notation that will be used throughout the paper. Let V denote the set of variables of interest. According to the graphical model paradigm, each variable is viewed as a vertex of a graph. Let r_v denote the number of values of variable $v \in V$, x_v a particular value of v, x_A an assignment to the variables in the subset A of V. To simplify notation x_V will be denoted by x.

We use trees as graphical representations for families of probability distributions over V that satisfy a common set of independence relationships encoded in the tree topology. In this representation, an edge of the tree shows a direct dependence, or, more precisely, the absence of an edge between two variables signifies that they are independent, conditioned on all the other variables in V. We shall call a graph that has no cycles a *tree*[1] and shall denote by E its edge set.

Now we define a probability distribution T that is *conformal* with a tree. Let us denote by T_{uv} and T_v the marginals of T:

$$\begin{aligned} T_{uv}(x_u, x_v) &= \sum_{x_{V-\{u,v\}}} T(x_u, x_v, x_{V-\{u,v\}}) \\ T_v(x_v) &= \sum_{x_{V-\{v\}}} T(x_v, x_{V-\{v\}}). \end{aligned}$$

Let $\deg v$ be the *degree* of vertex v, e.g. the number of edges incident to $v \in V$. Then, the distribution T is

[1] In the graph theory literature, our definition corresponds to a *forest*. The connected components of a forest are called trees.

conformal with the tree (V, E) if it can be factorized as:

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

The distribution itself will be called a tree when no confusion is possible. If the tree is connected, e.g. it *spans* all the nodes in V, it is often called a *spanning tree*.

An equivalent representation for T in terms of conditional probabilities is

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

The form (2) can be obtained from (1) by choosing an arbitrary root in each connected component and recursively substituting $\frac{T_{v\mathrm{pa}(v)}}{T_{\mathrm{pa}(v)}}$ by $T_{v|\mathrm{pa}(v)}$ starting from the root. $\mathrm{pa}(v)$ represents the parent of v in the thus directed tree or the empty set if v is the root of a connected component. The directed tree representation has the advantage of having independent parameters. The total number of free parameters in either representation is

$$\sum_{(u,v)\in E_T} r_u r_v - \sum_{v\in V} (\mathrm{deg} v - 1) r_v.$$

From the second representation of T one can notice that for distribution that is conformal with a tree, all the separation properties induced by the tree structure have a corresponding conditional independence property of the distribution. More concretely, in a tree there is at most one path between every two vertices. If node w is on the path between u and v we say that w *separates* u and v. Correspondingly, if we condition on w, the probability distribution T can be decomposed into the product of two factors, each one containing one of the variables u, v. Hence, u and v are independent given w.

Now we define a mixture of trees to be a distribution of the form

$$Q(x) = \sum_{k=1}^{m} \lambda_k T^k(x) \quad (3)$$

with

$$\lambda_k \geq 0,\ k = 1, \ldots, m; \qquad \sum_{k=1}^{m} \lambda_k = 1. \quad (4)$$

The tree distributions T^k are the *mixture components* and λ_k are called *mixture coefficients*. From the graphical models perspective, a mixture of trees can be viewed as a containing an unobserved choice variable z, taking value $k \in \{1, \ldots m\}$ with probability λ_k. Conditioned on the value of z the distribution of the visible variables x is a tree. The m trees may have different structures and different parameters. Note that because of the structure variable, a mixture of trees is not properly a belief network, but most of the results here owe to the belief network perspective.

3 THE BASIC ALGORITHM: ML FITTING OF MIXTURES OF TREES

This section will show how a mixture of trees can be fit to an observed dataset in the Maximum Likelihood paradigm via the EM algorithm [3]. The observations are denoted by $\{x^1, x^2, \ldots, x^N\}$; the corresponding values of the structure variable are $\{z^i,\ i = 1, \ldots N\}$.

Following a usual EM procedure for mixtures, the Expectation (E) step consists in estimating the posterior probability of each mixture component (tree) to generate datapoint x^i

$$Pr[z^i = k | x^{1,\ldots N}, model] = \gamma_k(i) = \frac{\lambda_k T^k(x^i)}{\sum_{k'} \lambda_{k'} T^{k'}(x^i)} \quad (5)$$

Then the expected complete log-likelihood to be maximized by the M step of the algorithm is

$$E[l_c \ | x^{1,\ldots N}, model] = \quad (6)$$
$$= \sum_{k=1}^{m} \Gamma_k [\log \lambda_k + \sum_{i=1}^{N} P^k(x^i) \log T^k(x^i)]$$
$$\Gamma_k = \sum_{i=1}^{N} \gamma_k(x^i), \quad k = 1, \ldots m \quad (7)$$
$$P^k(x^i) = \gamma_k(i)/\Gamma_k. \quad (8)$$

The sums Γ_k represent the total number of points assigned to each tree. By normalizing the posteriors $\gamma_k(i)$ with Γ_k we obtain a probability distribution P^k over the data set. To obtain the new distributions T^k, we have to maximize for each k the expression that is the negative of the crossentropy between P^k and T^k.

$$\sum_{i=1}^{N} P^k(x^i) \log T^k(x^i) \quad (9)$$

This problem can be solved exactly as shown in [2]. Here we will give a brief description of the procedure. First, one has to compute the mutual information between each pair of variables in V under the target distribution P

$$I_{uv} = \sum_{x_u x_v} P_{uv}(x_u, x_v) \log \frac{P_{uv}(x_u, x_v)}{P_u(x_u) P_v(x_v)}, \quad u, v \in V, u \neq v. \quad (10)$$

Second, the optimal tree structure E_T is found by a *Maximum Weight Spanning Tree* (MWST) algorithm using I_{uv} as the weight for edge $(u, v), \forall u, v \in V$. Once the tree is found, its marginals T_{uv} (or $T_{u|v}$), $(u, v) \in E_T$ are exactly equal to the corresponding marginals P_{uv} of the target distribution P. They are already computed as an intermediate step in the computation of the mutual informations I_{uv} (10).

In our case, the target distribution for T^k is represented by the posterior sample distribution P^k. Note

Figure 1: **The Basic Algorithm: ML Fitting of a Mixture of Trees**

Input: Dataset $\{x^1, \ldots x^N\}$

Initial model $m,\ T^k,\ \lambda^k,\ k = 1, \ldots m$

Procedure MWST(weights) that fits a maximum weight spanning tree over V

Iterate until convergence:

E step: compute $\gamma_k^i,\ P^k(x^i)$ for $k = 1, \ldots m,\ i = 1, \ldots N$ by (5), (7), (8)

M step:

M1. $\lambda_k \leftarrow \Gamma_k / N,\ k = 1, \ldots m$

M2. compute marginals $P_v^k,\ P_{uv}^k,\ u, v \in V,\ k = 1, \ldots m$

M3. compute mutual information $I_{uv}^k u, v \in V,\ k = 1, \ldots m$

M4. call MWST($\{ I_{uv}^k \}$) to generate E_{T^k} for $k = 1, \ldots m$

M5. $T_{uv}^k \leftarrow P_{uv}^k,\ ; T_v^k \leftarrow P_v^k$ for $(u,v) \in E_{T^k},\ k = 1, \ldots m$

that although each tree fit to P^k is optimal, for the encompassing problem of fitting a mixture of trees to a sample distribution only a local optimum is guaranteed to be reached. To complete the M step, one computes the maximizing values for the parameters λ by

$$\lambda_k^{new} \;=\; \Gamma_k / N.$$

The algorithm is summarized in figure 1. This procedure is based on one important assumption that should be made explicit now. It is the **Parameter independence assumption**: *The distribution $T^k_{v|\mathrm{pa}(v)}$ for any k, v and value of $pa(v)$ is a multinomial with $r_v - 1$ free parameters that are independent of any other parameters of the mixture.*

Shared structure It is possible to constrain the m trees to share the same structure, thus constructing a truly Bayesian network. To achieve this, it is sufficient to replace the weights in step **M4** of the basic algorithm by $\sum_k \Gamma_k I_{uv}^k$ and run the MWST algorithm only once to obtain the common structure E_T. The tree stuctures obtained by the basic algorithm are connected. The following section will give reasons and ways to obtain disconnected tree structures.

Missing variables are handled elegantly by trees. Any number of nonadjacent missing variables can be marginalized out in $\mathcal{O}(\max_v r_v)$ time and this bound grows exponentially with l, the size of the largest connected subset of missing variables.

Observed but unknown structure variable An interesting special case is the situation when the structure variable is in fact one of the observed variables (or a small subset thereof), but we don't know which one? To discover it, one can either: build several mixtures by conditioning on each one of the observables and then compare their posteriors, or: build one standard mixture model and then compare the mutual information between the structure variable and each of the others to identify the most likely candidate.

4 MAP MIXTURES OF TREES

In the previous section we have shown how to fit the ML mixture of spanning trees to a set of observations using the EM algorithm. Now we will extend the above procedure to the broader problem of finding the Maximum a Posteriori (MAP) probability mixture of trees for a given dataset. In other words, we will consider a nonuniform prior $P[model]$ and will be searching for the mixture of trees that maximizes

$$\begin{aligned} \log P[model|x^{1,\ldots N}] \;&=\; \log P[x^{1,\ldots N}|model] \\ &+\; \log P[model] + \text{constant.} \end{aligned} \tag{11}$$

Factorized priors The present maximization problem differs from the ML problem solved in the previous section only by the addition of the term $\log P[model]$. We can as well approach it from the EM point of view, by iteratively maximizing

$$\begin{aligned} E\left[\log P[model|x^{1,\ldots N}, z^{1,\ldots N}]\right] \;&=\; \log P[model] \\ &+\; E[l_c(x^{1,\ldots N}, z^{1,\ldots N}|model)] \end{aligned} \tag{12}$$

It is easy to see that the added term does not have any influence on the E step,which will proceed exactly as before. However, in the M step, we must be able to successfully maximize the r.h.s. of (12). In usual EM application this is enabled by the fact that we obtain a separate set of equations for the parameters of each mixture component. Therefore, we will look for priors over the trees parameters that also satisfy this decomposition. They are of the form

$$P[model] \;=\; P[\lambda_{1,\ldots m}] \prod_{k=1}^{m} P[T_k] \tag{13}$$

This class of priors is in agreement with the parameter independence assumption and includes the conjugate prior for the multinomial distribution which is the Dirichlet prior. A Dirichlet prior over a tree can be represented as a table of fictitious marginal probabilities P'^k_{uv} for each pair u, v of variables plus an *equivalent sample size* N' that gives the strength of the prior [8]. It is straightforward to maximize the a-posteriori probability of a tree: one has to replace the marginals P^k_{uv} in step **M2** by

$$\tilde{P^k_{uv}} \;=\; (N P^k_{uv} \,+\, N' P'^k_{uv}) / (N \,+\, N') \tag{14}$$

The above formula is obtained applying the derivation in [8] to the special case of a tree distribution and considering the parametrization of the Dirichlet distribution for which the mode is equal to the mean. Hence, the parameter estimates obtained in the M step will also represent mean values of the parameters under the (Dirichlet) posterior distribution.

A Dirichlet prior implies the knowledge of detailed prior information about the model. In particular it implies that the number of mixture components m is known. When this is not the case, but there is information about the marginal relations between the variables one can introduce it in the form of one table of fictitious marginals and and an equivalent sample size N'. From it one can create a fictitious dataset of size N' to augment the true training set. Then, the training should proceed just like for an ordinary ML model fitting.

MDL (Minimum Description Length) priors are even less informative priors. They attempt to balance the number of parameters that are estimated with the amount of data available, usually by introducing a penalty on model complexity. For mixtures of trees one can proceed in two fashions, differing on whether they maintain or drop the parameter independence. First we will describe methods to reduce the number of parameters while keeping them independent.

Edge pruning and prior on m. To control the number of components m, one can introduce a prior $P[m]$ and compare model posteriors obtained from (11). To penalize the number of parameters in each component notice that adding a link (u, v) in a tree contributes $\Delta_{uv} = (r_u - 1)(r_v - 1)$ parameters w.r.t. a factorized distribution. One can also choose a uniform penalty $\Delta_{uv} = 1$. Introducing a prior $P[T] \propto \exp\left[-\beta \sum_{uv\in E_T} \Delta_{uv}\right]$ is equivalent to maximizing the following expression for each mixture component (the mixture index k being dropped for simplicity)

$$\operatorname*{argmax}_{E_T} \sum_{uv\in E_T} [\Gamma I_{uv} - \beta\Delta_{uv}] \;=\; \operatorname*{argmax}_{E_T} \sum_{uv\in E_T} W_{uv} \tag{15}$$

To achieve this for any choice of Δ_{uv} it suffices to replace the weights in step **M4** by W^k_{uv} and to modify the MWST procedure so as to consider only positive weight edges. This prior is a factorized prior as well[2].

Smoothing (or regularization) methods consider one comprehensive model class (full spanning trees and a sufficiently large m) and within it introduce a bias towards a small *effective number of parameters*. Here we discuss a few techniques that can be applied to trees and direct the reader to consult the vast existing literature related to smoothing in clustering and discrete probability estimation for futher information on this subject.

• **Penalizing the entropy of the structure variable** by introducing the penalty term $-\alpha H(\lambda_{1,\ldots m})$. In this case, the λ_k cease being decoupled, and the resulting system of equations has to be solved numerically.

• **Smoothing with the marginal**. One computes the pairwise marginals for the whole dataset P^{total}_{uv} and replaces the marginals P^k_{uv} by

$$\tilde{P}^k_{uv} \;=\; (1-\alpha)P^k_{uv} + \alpha P^{total}_{uv}, \quad 0 < \alpha < 1 \tag{16}$$

This method and several variations thereof are discussed in [11]. Its effect is to give a small probability weight to unseen instances and to draw the components closer to each other, thereby reducing the effective value of m. For the method to be effective in practice α is usually a function of Γ_k and P^k_{uv}.

5 EXPERIMENTAL RESULTS

We have tested our model and algorithms for their ability to retrieve the dependency structure in the data, as density estimators and as classifiers.

For the first objective, we sampled 30,000 datapoints from a mixture of 5 trees over 30 variables with $r_v = 4$ for all vertices. All the other parameters of the generating model and the initial points for the algorithm were picked at random. The results on retrieving the original trees were excellent: out of 10 trials, the algorithm failed to retrieve correctly only 1 tree in 1 trial. This bad result can be accounted for by sampling noise. The tree that wasn't recovered had a λ of only 0.02. Instead of recovering the missing tree, the algorithm fit two identical trees to the generating tree with the highest λ. The difference between the log likelihood of the samples of the generating model and the approximating model was 0.41 bits per example. On all the correctly recovered trees, the approximating mixture had a higher log likelihood for the sample set than the generating distribution.

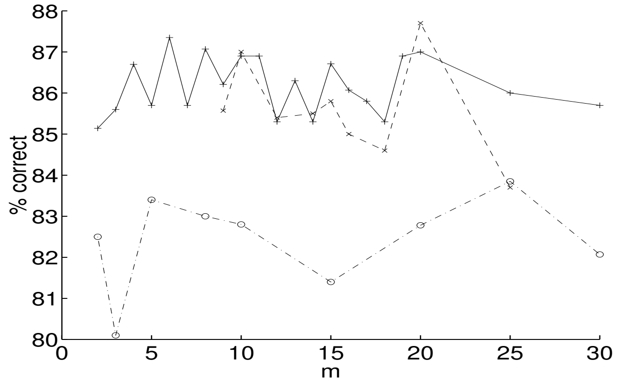
We also tested the basic algorithm as density estimator by running it on a subset of binary vector representations of handwritten digits and measuring the compression rate. The datasets consist of normalized and quantized 8x8 binary images of handwritten digits made available by the US Postal Service Office for Advanced Technology. One dataset contained images of single digits in 64 dimensions, the second contained 128 dimensional vectors representing randomly paired digit images. The training, validation and test set contained 6000, 2000, and 5000 exemplars respectively. The data sets, the training conditions and the algorithms we compared with are described in [4]. We tried mixtures of 16, 32, 64 and 128 trees, fitted by the basic algorithm. The training set was used to fit the model parameters and the validation set to determine when EM has converged. The EM iteration was stopped after the first decrease in the log-likelihood on the validation set. For each of the two datasets we chose the mixture model with the highest log-likelihood on the validation set and using it we calculated the average log-likelihood over the test set

[2] Note that to use $P[m]$ together with edge pruning one has to compute the normalization constant in (11).

Table 1: Compression rates (bits per digit) for the single digit (Digit) and double digit (Pairs) datasets. Boldface marks the best performance on each dataset.

m	Digits				Pairs			
Run #	1	2	3	avg.	1	2	3	avg.
16	34.9810	34.4443	34.7127	34.7108	79.4694	79.0958	79.2068	79.2602
32	34.5525	34.5785	34.3131	**34.4478**	78.9818	79.0193	78.9861	**78.9891**
64	34.6896	35.1253	34.7343	34.8616	79.6093	79.8907	79.6397	79.7097
128	34.7459	34.8440	35.0777	34.9167	81.1271	81.1314	81.5354	81.3818

Figure 3: Performance of different algorithms on the Australian Credit dataset. – is mixture of trees with $\beta = 10$, - - is mixture of trees with $\beta = 1/m$, -·- is mixture of factorial distributions.



(in bits per example). These results are shown in figure 2. The other algorithms mentioned in the table are the mixture of factorial distributions (MF), the completely factorized model (which assumes that every variable is independent of all the others) called "Base rate" (BR), the Helmholtz Machine trained by the wake-sleep algorithm [4] (HWS), the same Helmholtz Machine where a mean field approximation was used for training (HMF) and a fully visible and fully connected sigmoid belief network (FV). Table 1 displays the performances of all the mixture of trees models that we tested.

The results are very good: the mixture of trees is the absolute winner for compressing the simple digits and comes in second as a model for pairs of digits. This suggests that our model (just like the mixture of factorized distributions) is able to perform good compression of the digit data but is unable to discover the independence in the double digit set. A comparison of particular interest is the comparison in performance between the mixture of trees and the mixture of factorized distribution. In spite of the structural similarities, the mixture of trees performs significantly better than the mixture of factorial distribution indicating that there exists some structure that is exploited by the mixture of spanning trees but can't be captured by a mixture of independent variable models.

6 Classification with mixtures of trees

Classification is an common an important task for which probabilistic models are used. Therefore, this section will be devoted to experimentally assessing the performance of the mixture of trees model in classification tasks.

For all the tasks, we trained one model on the whole training set, treating the class variable like any other variable. In the testing phase, a new instance was classified by picking the most likely value of the class variable given the other variables' settings.

We investigated the performance of mixtures of trees on four classification tasks from the UCI repository [1]. In the first experiment, the data set was the Australian database [1]. It has 690 examples each consisting of 14 attributes and a binary class variable. Six of the attributes (numbers 2, 3, 7, 10, 13 and 14) were real-valued and they were discretized into 4 (for attribute 10) respectively 5 (for all other attributes) roughly equally sized bins. In our experiments, in order to compare with [9] the test and training set sizes were 70 and 620 respectively (a ratio of roughly 1/9). For each value of m that was tested 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. In a first series of runs all training parameters were fixed except for m. In the second series of runs, we let β, the edge pruning parameter, change in such a way as to keep $m\beta$ approximatively constant. This results in roughly the same fraction of pruned edges independently of the number of components. The results, which are slightly better than in the previous experiment, are presented in figure 3. What is common to both series of experiments is the relatively large range of values of m for which the performance stays close to its top. We hypothesize that this is caused by the multiple ways the models are smoothed: edge pruning, smoothing with the marginals and early stopping.

The best performance of the mixtures of trees in the second case is compared to other published results for the same dataset in table 2. For comparison, correct classification rates obtained and cited in [9] on training/test sets of the same size are: 87.2% for mixtures of

Figure 2: Compression rates (bits per digit) for the single digit (•) and double digit (■) datasets. MST is a mixture of spanning trees, MF is a mixture of factorial distributions, BR is the base rate model, HWS is a Helmholtz machine trained by the Wake-Sleep algorithm, HMF is a Helmholtz machine trained using the Mean Field approximation, FV is fully visible fully connected sigmoidal Bayes net.

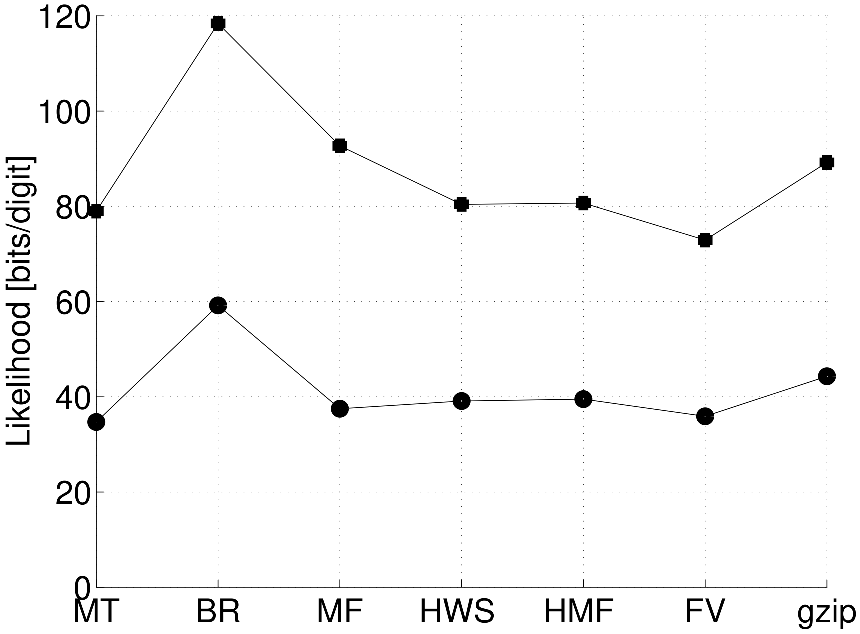


Table 2: Performance comparison between the mixture of trees model and other classification methods on the AUSTRALIAN dataset. For mixtures of trees, we report the best result only. The results for mixtures of factorial distribution are those reported in [9]. All the other results are from [10].

Method	% correct	Method	% correct
Mixture of trees $m = 20$, $\beta = 4$	**87.8**	Backprop	84.6
Mixture of factorial distributions (D-SIDE in [9])	87.2	C4.5	84.6
Cal5	86.9	SMART	84.2
ITrule	86.3	Bayes Trees	82.9
Logistic discrimination	85.9	K-nearest neighbor	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

Table 3: Performance of mixture of trees models on the MUSHROOM dataset. m=10 for all models.

Algorithm	Correctly classified	Soft class $\sum \gamma_{\text{true}}/N$	Test compression (bits/datapoint)	Train compression (bits/datapoint)
No smoothing	.998	.997	27.63	27.09
Smooth w/ marginal $\alpha_M = 0.3,\ \alpha_P = 20$	1	.9999	27.03	26.97
Smooth w/ uniform $\alpha_M = 0.3,\ \alpha_P = 200$	1	.9997	27.39	27.02

factorial distributions and 86.9% for the next best model (a decision tree called Cal5). The full description of the other methods can be found in [9, 10]. It can be seen that on this dataset the mixture of trees achieves the best performance, followed by the mixture of factorial distributions from [9].

The second data set used was the MUSHROOM database [15]. This data set has 8124 instances of 23 discrete attributes (including the class variable, which is treated like any other attribute for the purpose of model learning). The training set comprised 6000 randomly chosen examples, and the test set was formed by the remaining 2124. The smoothing methods used were a) a penalty α_P on the entropy of the mixture variable and b) smoothing with the marginal according to (16) or similarly with a uniform distribution. The smoothing coefficient α_M was divided between the mixture components proportionally to $1/\Gamma_k$. For this dataset, smoothing was effective both in reducing overfitting and in improving classification performance. The results are shown in table 3. The soft classification colums expresses an intergrated measure of the confidence of the classifier. It is visible that besides the classification being correct, the classifier also has achieved high confidence.

The last task was the classificaton of DNA SPLICE-junctions. The domain consists of 60 variables , representing a sequence of DNA bases, and an additional class variable. The task is to determine if the middle of the sequence is a splice junction and what is its type[3]. 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 coded here by the symbols (C, A, G, T). The data set consists of 31175 labeled examples. We compared the Mixture of trees model with two categories of classifiers and thus we performed two series of experiments. A third experiment involving the SPLICE data set will be described later.

For the first series, we compared our model's performance against the reported results of [16] and [12] who used multilayer neural networks and knowledge-based neural networks for the same task. The sizes of the training set and of the test set are reproduced from the above cited papers; they are 2000 and 1175 examples respectively. We constructed trees ($m = 1$) and mixtures of $m = 3$ trees with different smoothing values α. Fitting the single tree can be done in 1 step. To fit the mixture, we separated N_{valid}=300 examples out of the training set and learned the model using the EM algorithm on the remaining 1700. The training was stopped when the] likelihood of the validation set stopped decreasing. This can be regarded as an a additonal smoothing for the $m = 3$ model. The results, averaged over 20 trials, are presented in figure 4a. It is visible that the tree and the mixture of trees model perform very similarly, with the single tree showing an insignificantly better classification accuracy. Since a single tree has about three times fewer parameters than a mixture with $m = 3$ we strongly prefer the former model for this task. Notice also 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" mixture of trees model ($\alpha = 100$) all the trees/mixture of trees models significantly outperform the other models tested on this problem. Note that whereas the mixture of trees contains no prior knowledge about the domain, the KBNN does.

The second set of experiments pursued a comparison with benchmark experiments on the SPLICE data set that are part of the DELVE data base [14]. The DELVE benchmark uses subsets of the SPLICE database with 100, 200 and 400 examples for training. Testing is done on 1500 examples in all cases. The algorithms tested by DELVE and their performances are shown in figure 4. We fitted single trees ($m = 1$) with different degrees of smoothing. We also learned Naive Bayes (NB) and Tree Augmented Naive Bayes (TANB) models [5]. The models are mixtures of factorial distributions and mixture of trees respectively where the choice variable is visible and represents the class variable. Thus a NB or a TANB model can be fitted in one step, just like a tree.According to the DELVE protocol, we run our algorithms 20 times with different random initializations on the same training and testing sets.

The results are presented in figures 4b,c and d. Most striking in these plots is the dramatic difference between the methods in DELVE and the classification obtained by a simple tree: in all cases the error rates of the tree models are *less than half* of the performance of the best model tested in DELVE! Moreover, the average error of a single tree trained on 400 examples is 5.5%, which is only 1.2% away from the average error of trees trained on the 2000 examples dataset. The explanation for this remarkable accuracy preservation with the decrease of the number of examples is discussed later in this section. The Naive Bayes model exhibits a behaviour that is very similar to the tree model and only slightly less accurate. However, augmenting the Naive Bayes model to a TANN signinficantly hurts the classification performance. The plots also allow us to observe the effect of the degree of smoothing when it varies from none to very large. In contrast to the previous experiment on SPLICE data, here smoothing has a benefic effect on the classification accuracy for values under a certain treshold; for larger values the accuracy is strongly degraded by smoothing. These accuracy profiles can be observed in tree models

[3]The DNA is composed of sections that are useful in coding proteins, called *exons*, and of inserted sections of noncoding material called *introns*. Splice junctions are junctions between an exon and an intron and they 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 places where the intron ends and the next exon, or coding section, begins.

Figure 4: Comparison of classification performance of the mixture of trees and other models on 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, Linear least squares, Linear least squares ensemble learning, ME (mixture of experts)-ensemble learning, ME-early stopping. TANB is the Tree Augmented Naive Bayes classifier of [5], NB is the Naive Bayes classifier, 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.

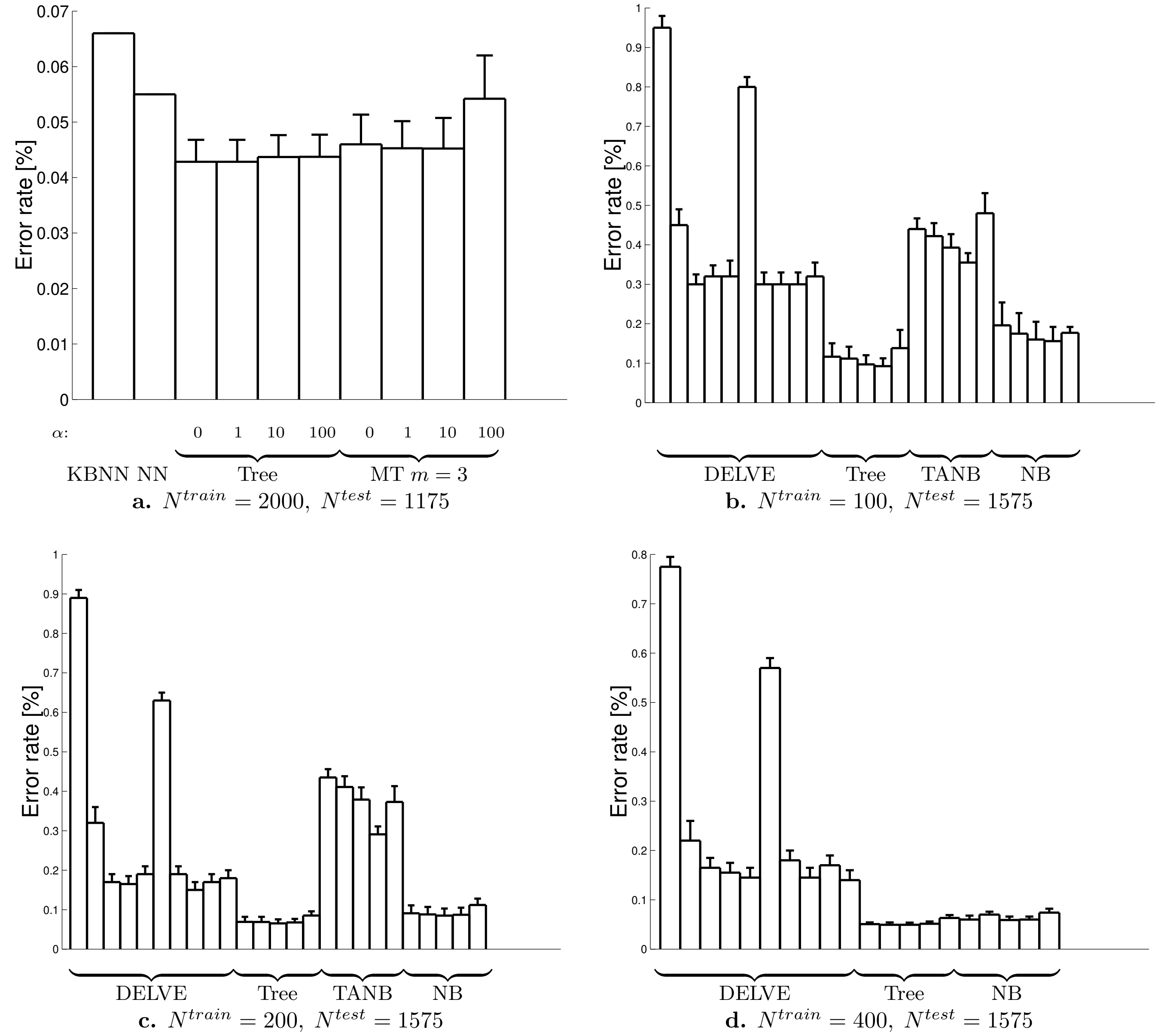


Figure 5: Cumulative incidence matrix of 20 trees fit to 2000 examples of the SPLICE data set with no smoothing. The size of the square at coordinates ij represents the number of trees (out of 20) that have an edge between variables i and j. No square means that this number is 0. Only the lower half 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.

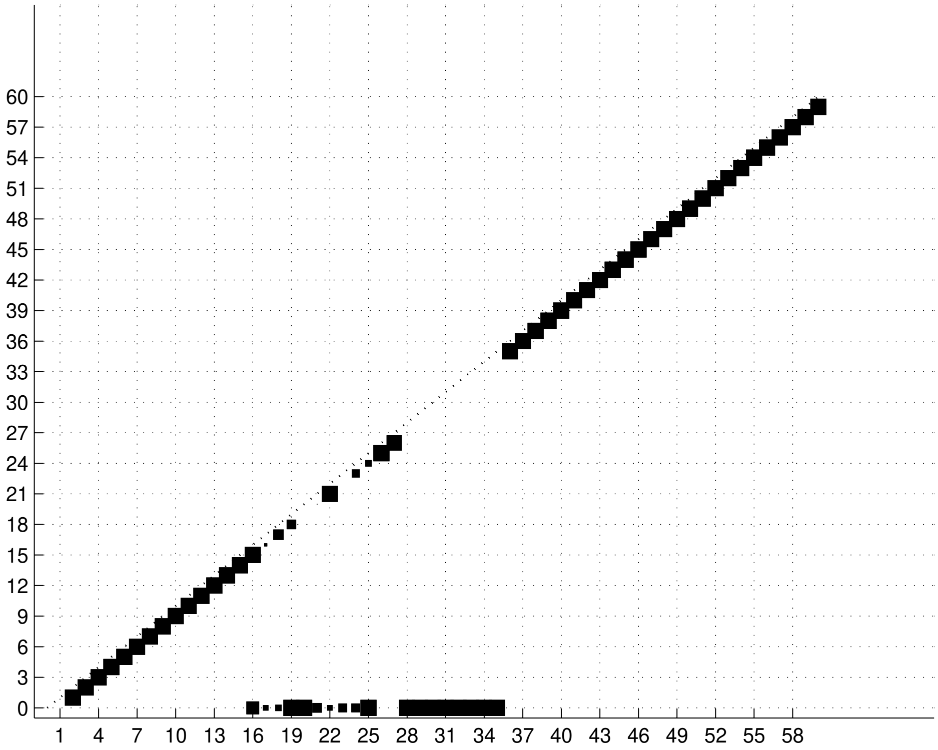


Figure 6: The encoding of the IE and EI splice junctions as discovered by the tree learning algorithm compared to the ones given in J.D. Watson & al., "Molecular Biology of the Gene"[17]. Positions in the sequence are consistent with our variable numbering: thus the splice junction is situated between 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 basis with a non-negligible probability.

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**

as wel as in the TANN and Naive Bayes models and they are similar to the bias-variance tradeoff curves commonly encountered in machine learning applications. Also not surprisingly, the effect diminishes with the increasing size of the data set (and is almost invisible for a training set size of 400).

Discovering structure. Figure 5 presents a summary of the tree structures learned from the N=2000 example set in the form of a cumulated incidence matrix. The incidence matrices of the 20 graph structures obtained in the experiment have been added up[4] 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 incidence matrix. No square means that the respective element is 0. Since the incidence matrix are symmetric, only half the matrix is shown. From figure 5 we can see first that the tree structure is extremely stable over the 20 trials. Variable 0 represents the class variable; the hypothetic 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 6. The same figure displays the "true" encodings of the IE and EI junctions as given in [17]. The ressemblance between the two encodings is almost perfect. Thus, we can conclude that for this domain, the tree model not only provides a good classifier but also discovers the true undelying model of the physical reality that generates the data! Remark also that the algorithm arrives at this result *without any prior knowledge*: it does not know which one is the class variable and it doesn't even know that 60 variables out of 61 are in a sequence.

The single tree classifier as an automatic feature selector. Let us examine the single tree classifier that was used for the SPLICE data set more closely. According to the separation 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 will be relevant for determinig it's 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 accurately fit 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.

Sensitivity to irrelevant features To verify that indeed the single tree classifier acts like a feature selector, we performed the following experiment, using again the SPLICE data. We augmented the variable set with another 60 variables, each taking 4 values with randomly and idependently assigned probabilities. The rest of the experimental conditions (training set, test set and number of random restarts) were identical to the first experiment. We fitted a set of models with $m = 1$ and no smoothing and compared its structure and performance to the corresponding model set in the first experiment series. The structure of the new models, in the form of a cumulative incidence matrix, is shown in figure 7. 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[5].

7 CONCLUSIONS

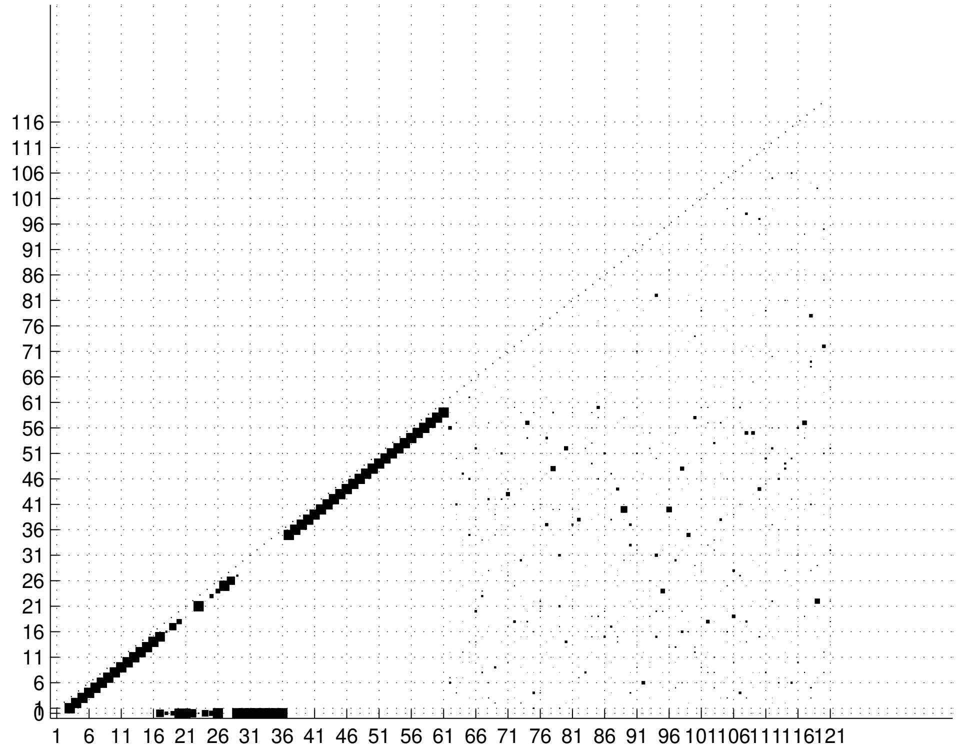
This paper has shown a way of modeling and exploiting sparse dependency structure that is conditioned on values of the data. Without literally being a belief net, the mixture of trees that we introduced, by playing on variable topology independencies, is one in spirit. Trees do not suffer from the exponential computation demands that plague both inference and structure finding in wider classes of belief nets. The algorithms presented here are linear in m and N and quadratic in $|V|$. The loss in modelling power is compensated by using mixtures instead of single trees. The possibility of pruning a mixture of trees can play a role in classification, as a means of automatically selecting the variables that are relevant for the task.

The importance of using the right priors in constructing models for real-world problems can hardly be underestimated. In this context, two issues arise: 1. how is it possible to devise good priors over the class of mixtures of trees models ? and 2. what is the computational burden involved in taking priors into account ? The present paper has offered partial answers to both these questions: it has presented a broad class of priors that are efficiently handled in the framework of our algorithm and

[4]The reader is reminded that the incidence matrix of a graph has a 1 in position ij if the graph has an edge connecting vertices i and j and a 0 otherwise.

[5]The standard deviation of the accuracy is 3.5% making this difference insignificant.

Figure 7: The cumulated incidence 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.



it has shown that this class includes important priors like the MDL prior and the Dirichlet prior.

Classification is one of the most common and important tasks that are required from automatically learned models. Therefore in our experiments it has received special attention. The experiments have demonstrated excellent classification performance for the mixture of trees model as well as for the single tree model. We have shown that, for the single tree classifier, the tree structure acts like an implicit feature selector, which gives this model good capabilities in the presence of large sets of irrelevant attributes.

ACKNOWLEDGEMENTS

Thanks to Brendan Frey for making the digits datasets available to us.

References

[1] U. C. Irvine Machine Learning Repository. ftp://ftp.ics.uci.edu/pub/machine-learning-databases/.

[2] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *"IEEE Transactions on Information Theory"*, IT-14(3):462–467, May 1968.

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

[4] Brendan J. Frey, Geoffrey E. Hinton, and Peter Dayan. Does the wake-sleep algorithm produce good density estimators? In D. Touretsky, M. Mozer, and M. Hasselmo, editors, *Neural Information Processing Systems*, number 8, pages 661–667. MIT Press, 1996.

[5] Nir Friedman and Moses Goldszmidt. Building classifiers using Bayesian networks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 96)*, pages 1277–1284, Menlo Park, CA, 1996. AAAI Press.

[6] Dan Geiger. An entropy-based learning algorithm of bayesian conditional trees. In *Proceedings of the 8th Conference on Uncertainty in AI*, pages 92–97. Morgan Kaufmann Publishers, 1992.

[7] B. German. Glass identification database. U.C. Irvine Machine Learning Repository.

[8] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: the com-

bination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[9] Petri Kontkanen, Petri Myllymaki, and Henry Tirri. Constructing bayesian finite mixture models by the EM algorithm. Technical Report C-1996-9, Univeristy of Helsinky, Department of Computer Science, 1996.

[10] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Publishers, 1994.

[11] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.

[12] Michiel O. Noordewier, Geoffrey G. Towell, and Jude W. Shawlik. Training Knowledge-Based Neural Networks to recognize genes in DNA sequences. In Richard P. Lippmann, John E. Moody, and David S. Touretztky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 530–538. Morgan Kaufmann Publishers, 1991.

[13] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman Publishers, San Mateo, CA, 1988.

[14] Carl E. Rasmussen, Radford M. Neal, Geoffrey E. Hinton, Drew van Camp, Michael Revow, Zoubin Ghahramani, R. Kustra, and Robert Tibshrani. *The DELVE Manual*. http://www.cs.utoronto.ca/ delve, 1996.

[15] Jeff Schlimmer. Mushroom database. U.C. Irvine Machine Learning Repository.

[16] Geoffrey Towell and Jude W. Shawlik. Interpretation of artificial neural networks: Mapping Knowledge Based Neural Networks into rules. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 9–17. Morgan Kaufmann Publishers, 1992.

[17] James D. Watson, Nancy H. Hopkins, Jeffrey W. Roberts, Joan Argetsinger Steitz, and Alan M. Weiner. *Molecular Biology of the Gene*, volume I. The Benjamin/Cummings Publishing Company, 4 edition, 1987.