Pp. 237–248 in *Probabilistic Reasoning and Bayesian Belief Networks*, edited by A.Gammerman. Alfred Waller Publishers, Suffolk 1995.

Constructing Computationally Efficient Bayesian Models via Unsupervised Clustering

Petri Myllymäki and Henry Tirri
University of Helsinki, Department of Computer Science*
P.O.Box 26, FIN-00014 University of Helsinki, FINLAND
Petri.Myllymaki@cs.Helsinki.FI, Henry.Tirri@cs.Helsinki.FI

Abstract

Given a set of samples of an unknown probability distribution, we study the problem of constructing a good approximative Bayesian network model of the probability distribution in question. This task can be viewed as a search problem, where the goal is to find a maximal probability network model, given the data. In this work, we do not make an attempt to learn arbitrarily complex multi-connected Bayesian network structures, since such resulting models can be unsuitable for practical purposes due to the exponential amount of time required for the reasoning task. Instead, we restrict ourselves to a special class of simple tree-structured Bayesian networks called Bayesian prototype trees, for which a polynomial time algorithm for Bayesian reasoning exists. We show how the probability of a given Bayesian prototype tree model can be evaluated, given the data, and how this evaluation criterion can be used in a stochastic simulated annealing algorithm for searching the model space. The simulated annealing algorithm provably finds the maximal probability model, provided that a sufficient amount of time is used.

1 Introduction

Learning structures from data is always relative to a class of models, set of structures which share some common properties. In our work we have restricted ourselves to learning probabilistic network structures, i.e., constructing a Bayesian network representation of a probability distribution described by a set of representative data samples. This learning task can be described in the abstract setting as a search in the network space, where the goal is to find a maximal probability network model for the data. The learning process has two important components: an evaluation criterion for a candidate network, and a search algorithm for selecting new candidate networks. Under certain assumptions, a probabilistic evaluation criterion for comparing different Bayesian network models can be constructed by computing directly the probability of a given Bayesian network [7], or indirectly [20] by using the Minimum Description Length approach [32]. This type of criteria are totally independent of the search algorithm used, and hence each search algorithm can be combined with several different criteria. In this paper, we use a combination of the techniques adopted from [20, 32] for constructing an example of suitable evaluation criteria.

In principle, the best model for the problem domain probability distribution can be found by going through all the possible networks, and choosing the model with the highest probability. Unfortunately, in the Bayesian network framework the number of possible models is too large for such an exhaustive search approach to be used in practice (see the combinatorial analysis in [33]). For this reason most existing learning methods for Bayesian networks typically use simple greedy heuristics for searching the model space. In order to improve the performance of the learning

*This research was supported by the Technology Development Center (TEKES).

algorithms, more elaborate search algorithms need to be used. Standard numerical optimization methods, such as gradient descent or conjugate gradient methods are not directly applicable since the gradient of the function to be maximized is not available. Another, more severe drawback with these methods is that there is nothing to prevent the methods from getting stuck in local maxima. This is an important aspect, as in the large model space in question the probability of having a large number of local maxima is very high, in which case the quality of the solution found with such optimization methods may be poor.

Simulated annealing [24] is an iterative stochastic sampling method which has been shown to be able to find a maximum probability state of the sampling space almost surely, provided that a sufficient number of iterations is used [11]. After being introduced to the optimization theory community in [14], simulated annealing has been applied to many different (NP-hard) optimization problems, such as TSP, graph partitioning, graph coloring, number set partitioning and clustering (for an extensive survey of applications, see [1, pp.89–90]). Unfortunately, the number of iterations for the theoretically guaranteed convergence is too high for practical purposes, but in many cases good results has been obtained even with a relatively small number of iterations [1].

Simulated annealing has also been used for implementing the reasoning computation in Bayesian networks [26]. However, as suggested in [5], in this work we use the simulated annealing algorithm for learning, i.e., finding a globally maximal probability Bayesian network model for our problem domain. In contrast to most existing learning algorithms for Bayesian networks, we do not try to learn arbitrarily complex multi-connected Bayesian network structures. In this restriction we are motivated by the fact that in the general case the resulting model can be unsuitable for practical purposes as the Bayesian reasoning task with multi-connected networks may take an exponential amount of time [6]. Instead, we restrict ourselves to a special class of simple tree-structured Bayesian networks, *Bayesian prototype trees* [28, 29], for which a polynomial time algorithm for Bayesian reasoning exists [31]. In addition, the Bayesian prototype tree model can also be implemented extremely efficiently on massively parallel hardware, as there is a direct mapping from a Bayesian tree structure to a multi-layer feedforward neural network structure [28]. The model also conforms interestingly to the intuitively appealing *memory-based reasoning* paradigm (see e.g. [15]), and it can be seen as a Bayesian solution to the case matching and case adaptation problems (see [29]) in such domains.

The Bayesian prototype tree model is based on the assumption that the problem domain probability distribution can be approximated by a set of prototype vectors, where each prototype represents a set of (in some sense) similar problem domain instances. The approach is very similar to finite mixture models, and statistical kernel-based density estimators, where the problem domain probability distribution is approximated as a finite sum of particular types of kernel distributions (e.g., Gaussian). Such models are becoming increasingly popular in the *neural network* community, since the models can often be implemented very efficiently on massively parallel neural network architectures [25, 37, 21, 13]. In our approach we restrict ourselves to discrete problem domains, which allows us to drop all assumptions about the form of the underlying probability distribution. The prototype vectors are regarded as values of a latent hidden random clustering variable, which forms the root of the prototype tree model, and the actual problem domain random variables are the leaves of the tree. The approach resembles the discrete version of the *AutoClass* model [4], the *simple Bayesian classifier* model in [18] and [19], and the finite mixture model of multivariate Bernoulli distributions in [12], but unlike these models, our network can be used also for general regression tasks, not only for classification.

The Bayesian prototype tree model is described in more detail in Section 2. Naturally the accuracy of the model depends on how well the chosen Bayesian prototypes reflect the actual probability distribution. In [28, 29] the prototypes were assumed to be provided by a domain expert. However, although using experts is viable in some domains, in many cases such prototypes need to be inferred from raw data. In Section 3, we determine the Bayesian prototype tree structure and the corresponding parameters, given a set of training data. A probabilistic evaluation criterion for comparing different prototype tree models is represented in Section 4. Using this evaluation criterion as an example, we show in Section 5 how to use the simulated annealing algorithm for

searching for the best model of the problem domain. Results of some preliminary experiments with this approach are reported in Section 6.

2 The Bayesian prototype tree model

We code the problem domain knowledge by using values of m discrete attributes (random variables) $A_1, \ldots, A_m$, where each attribute A_i has n_i possible values, $a_{i1}, \ldots, a_{in_i}$. Hence the problem domain space consists of $\prod_{i=1}^{m} n_i$ possible data vectors $\vec{d}_h$, $\vec{d}_h = (V_h(A_1), \ldots, V_h(A_m))$, where $V_h(A_i)$ is the value of attribute A_i in instance $\vec{d}_h$, $V_h(A_i) \in \{a_{i1}, \ldots, a_{in_i}\}$.

The Bayesian prototype tree model is based on the assumption that the data vectors $\vec{d}_h$ in the problem domain space are not distributed uniformly, but form *clusters* or *classes* of (in some sense) similar vectors. This is a reasonable assumption in many real-world domains, although in certain more or less artificial (logical) problems this is not necessarily the case. Let $\mathcal{C}$ denote a random variable the values of which represent the problem domain classes, and let $\mathcal{C}_1, \ldots, \mathcal{C}_l$ be the labels of the l existing classes. The probability distribution inside each class $\mathcal{C}_k$ is represented by a *class prototype vector*

$$\vec{P}_k = (P_k(a_{11}), \ldots, P_k(a_{1n_1}), \ldots, P_k(a_{m1}), \ldots, P_k(a_{mn_m})),$$

where $P_k(a_{ij})$ expresses the likelihood for attribute A_i to have value a_{ij} in class $\mathcal{C}_k$:

$$P_k(a_{ij}) = P(A_i = a_{ij} | \mathcal{C} = \mathcal{C}_k).$$

If the data vectors can be partitioned to form a set of separate classes where the members of each class are relatively close to each other, knowing membership in a class (say, $\mathcal{C}_k$) is sufficient information for determining the values of the attributes, since they are now given by the corresponding class prototype vector $\vec{P}_k$. In the Bayesian framework this means that *all the variables A_i are conditionally independent of each other, given the value of the variable $\mathcal{C}$* (which does not, however, mean that the attributes are independent). In other words, the problem domain probability distribution can be modeled by a tree-structured Bayesian network, where a single variable $\mathcal{C}$ is the root of the tree, and variables A_i form the leaves (see Figure 1).

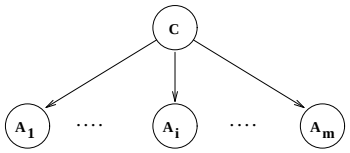

Figure 1: Bayesian prototype tree representation of the problem domain probability distribution with a class variable $\mathcal{C}$ and attribute variables $A_1, \ldots, A_m$.

It should be noted that the Bayesian prototype tree model does not impose any unrealistic assumptions about independencies between the measurable attributes $A_1, \ldots, A_m$, since by introducing an artificial (hidden) clustering variable $\mathcal{C}$ and making the corresponding *local* conditional independence assumptions inside each class, we do not have to care about any possible *global* dependencies between any two measurable attributes. Naturally, the goodness of our model depends on how well the classes reflect the actual clusters in the real world, but theoretically the Bayesian prototype tree model can be used to represent any probability distribution. As an extreme example, consider the case where each of the data vectors $\vec{d}$ is modeled by a separate single-member class: this kind of model can certainly be used to represent any probability distribution exactly. Naturally the huge number of classes makes this primitive approach infeasible, but if the data are not completely uniformly distributed, classes with similar vectors can be combined to decrease

the size of the model. An additional problem is of course, that the problem domain probability distribution is not usually fully given, but only represented by a set of sample vectors. In the following, we show how to construct a Bayesian prototype tree model, given a set of sample data and a partitioning of the sample vectors.

3 Constructing prototype trees from data

Let our *training data set* $\mathcal{D}$ consist of N complete[1] data vectors $\vec{d}_1, \ldots, \vec{d}_N$. We start by assuming that l, the number of clusters in the problem domain, does not exceed N, the number of items in the training set. This is usually a safe assumption to make, since otherwise it can be argued that as there are clusters in the domain space with no representatives in the training set, the size of the training set is too small for successful construction of a problem domain model.

Let us initially consider clustering the N data vectors of the training set into N clusters $\mathcal{C}_1, \ldots, \mathcal{C}_N$, and provide each data vector $\vec{d}_h$ with the corresponding cluster index $c(h) \in \{1, \ldots, N\}$. The *partition vector* $\vec{c} = (c(1), \ldots, c(N))$, consisting of values of all the N data cluster indices, represents one possible partitioning of the data into at most N classes. The actual number of classes, l, is the number of non-empty clusters, where a cluster $\mathcal{C}_k$ is empty if $c(h) \neq k$, for all $h = 1, \ldots, N$.

Maximum likelihood estimates of the class prototype vectors corresponding to a given partitioning of the training data into l classes can be either derived by simply using the relative frequencies of value occurrences, or as noted in [7, 18], approximated more accurately with the formula

$$P_k(a_{ij}) = \frac{F_k(a_{ij}) + 1}{|\mathcal{C}_k| + n_i}, \tag{1}$$

where $F_k(a_{ij})$ is the number of vectors in cluster $\mathcal{C}_k$ with the property $A_i = a_{ij}$, and $|\mathcal{C}_k|$ is the total number of vectors partitioned in cluster $\mathcal{C}_k$. In addition, to be able to use our prototype tree model for Bayesian reasoning, we need also l prior probabilities, one for each of the classes $\mathcal{C}_k$. As the conditional probabilities in (1), these can be approximated by

$$P(\mathcal{C}_k) = \frac{|\mathcal{C}_k| + 1}{N + l}. \tag{2}$$

As in our framework a given data partition $\vec{c}$ fully determines (through formulae (1) and (2)) the corresponding Bayesian prototype tree model, we can identify each model with the corresponding partition vector, and represent each $\mathcal{M}$ as a vector, $\mathcal{M} = (c(1), \ldots, c(N))$. Hence to find an optimal Bayesian prototype tree model, we need to find an optimal partition vector among the N^N different vectors. In Section 5 we show how the simulated annealing algorithm can be used for searching through the space of partition vectors. To be able to use the search algorithm, we need an evaluation criterion for comparing alternative clusterings. In the next section, we present one possible criterion based on probability theory. Observe that in our Bayesian framework the evaluation criterion and the actual search algorithm used for finding the most probable model are completely independent of each other, and hence e.g. the simulated annealing approach can be used with various different evaluation criteria, not only with the one example given below, and correspondingly, each criterion can be used with several search algorithms.

4 An evaluation criterion for prototype trees

Let $\mathcal{M}$ denote a Bayesian prototype tree constructed as described in the previous section using a data set $\mathcal{D}$. Assuming that the training set $\mathcal{D}$ represents the problem domain distribution well, an optimal prototype tree is obtained by finding a model which maximizes the probability

$$P(\mathcal{M} \mid \mathcal{D}) = \frac{P(\mathcal{M})P(\mathcal{D} \mid \mathcal{M})}{P(\mathcal{D})}.$$

[1] Incomplete data vectors can be transformed to complete vectors by adding an 'unknown' category to the value sets of the attributes with missing values [4].

As the probability $P(\mathcal{D})$ is constant for a given training set, we can compare the likelihood of different prototype sets by using as the criterion the formula

$$P(\mathcal{M} \mid \mathcal{D}) = cP(\mathcal{M})P(\mathcal{D} \mid \mathcal{M}), \tag{3}$$

where c can be any positive constant. This criterion can also be approximated using the *Minimum Description Length (MDL)* [32] approach, where the goal is to minimize the size S of the model $\mathcal{M}$, given the data:

$$S(\mathcal{M} \mid \mathcal{D}) = S(\mathcal{M}) + S(\mathcal{D} \mid \mathcal{M}) = -\log P(\mathcal{M}) - \log P(\mathcal{D} \mid \mathcal{M}).$$

The first term $S(\mathcal{M})$ is the size of the prototype tree structure (with the corresponding parameters $P_k(a_{ij})$ and $P(\mathcal{C}_k)$), and the term $S(\mathcal{D} \mid \mathcal{M})$ can be regarded as the size of the *error* of the model $\mathcal{M}$. To minimize the total model size, one has to deal with a tradeoff between these two terms: if the accuracy of the model is increased, this usually requires also increasing the size (the number of classes) of the model, whereas using a small size model usually causes the size of the error to increase. In this framework, the difficult problem of overfitting can be avoided, since the MDL principle gives a penalty to complex models representing the training data too accurately.

There are many different approaches to approximating the terms $S(\mathcal{M})$ and $S(\mathcal{D} \mid \mathcal{M})$. For example, we can apply Theorem 2.2 in [20] to derive an approximation for $S(\mathcal{D} \mid \mathcal{M})$ as

$$\begin{aligned} S(\mathcal{D} \mid \mathcal{M}) \;=\; & -N \sum_{k=1}^{l} \sum_{i=1}^{m} \sum_{j=1}^{n_i} P_k(a_{ij}) P(\mathcal{C}_k) \log \frac{P_k(a_{ij})}{P(a_{ij})} - N \sum_{k=1}^{l} P(\mathcal{C}_k) \log P(\mathcal{C}_k) \\ & -N \sum_{i=1}^{m} \sum_{j=1}^{n_i} P(a_{ij}) \log P(a_{ij}), \end{aligned} \tag{4}$$

where m is the number of attributes, n_i is the number of values of attribute A_i, l is the number of classes, N is the size of the training set $\mathcal{D}$, and the parameters $P_k(a_{ij})$ and $P(\mathcal{C}_k)$ are computed from the data as shown in Section 3. The prior probabilities $P(a_{ij})$ can be computed in a similar manner. The third term containing a sum of these prior probabilities is constant for a given data set, and can hence be ignored. An alternative criterion, which computes the probability $P(\mathcal{D} \mid \mathcal{M})$ directly, is given in [7].

To approximate the model size $S(\mathcal{M})$, let us first code all the different Bayesian trees using the information theoretically optimal coding scheme in [32]. As our model contains l prototype vectors and l prior probabilities (one for each class $\mathcal{C}_k$), we now need

$$S(\mathcal{M}) = l \sum_{i=1}^{m} n_i r + lr, \tag{5}$$

bits to code all the parameters in the model, where r is the number of bits needed to store a truncated real value. Observe that for a particular problem domain, m, n_i and r will be constants, but l can vary from 1 to N (the size of the data set). This is not the only encoding scheme possible, but it is straightforward and reflects well the intuitive bias towards "simpler" trees, i.e., trees with the root node having less possible values.

Combining the results above, we get

$$P(\mathcal{M} \mid \mathcal{D}) \propto 2^{-S(\mathcal{M}|\mathcal{D})} = 2^{-S(\mathcal{M})-S(\mathcal{D}|\mathcal{M})}, \tag{6}$$

where $S(\mathcal{M})$ is the size of the Bayesian tree structure as defined in (5), and $S(\mathcal{D} \mid \mathcal{M})$ is computed using formula (4).

5 Searching the prototype tree model space by simulated annealing

Given a Bayesian tree with l prototypes, and a partition of the data $\mathcal{D}$ into these l classes, we can use the criterion (6) for computing the probability of the given Bayesian tree structure, given the

data, and hence evaluate the goodness of the tree representation. However, for a Bayesian tree with l prototypes, there are l^N ways to cluster the data, if the size of the data set is N, so it is clearly computationally infeasible to use the proposed scheme for evaluating all the possible tree structures with all the possible data partitions. In [30] we presented a greedy heuristic for the search process. This heuristic was a "bottom-up" approach which starts by partitioning all the data vectors in separate clusters, and continues by combining the two prototypes which produce the greatest increase in the probability $P(\mathcal{M} \mid \mathcal{D})$. In the following, we show how the simulated annealing algorithm can be used for performing a more elaborate stochastic local search in the model space. Our approach is supported by the results of applying simulated annealing for finding globally optimal clusterings [3, 35, 16]. An alternative approach to our clustering problem can be obtained by using a family of algorithms based on alternating between finding the maximum probability model for a given clustering, and computing a new clustering based on the assumption that the current model is correct [9, 8]. Perhaps the most famous variant of this procedure is the computationally simple K-means clustering algorithm [22]. With this kind of approach, there would be no need for an evaluation criterion at all, as clustering is based on comparing data vectors against prototype vectors, not comparing models against each other. However, although the method converges provably under certain conditions to a locally optimal clustering [8, 36], the outcome depends on the number of the prototype vectors used, and the choice of the initial prototype vectors, and hence finding a globally optimal clustering with this approach is difficult. Some attempts towards more robust variations of the approach are reported in [17, 23], but the global convergence properties of the resulting algorithms are hard to analyze theoretically.

In simulated annealing, our clustering problem is solved by introducing a stochastic Markov chain process $\{\mathcal{M}_t \mid t = 0, 1, \ldots\}$, where X is a random variable the values of which are the partition vectors, and $\mathcal{M}_t$ is the value of X at time t. In the following, we show how to construct a process which converges to the optimal model $\mathcal{M}_*$ almost surely:

$$\lim_{t \to \infty} P(\mathcal{M}_t = \mathcal{M}_*) = 1. \tag{7}$$

Let $\mathcal{M}_t = (c_t(1), \ldots, c_t(N))$ be the value of random variable X at time t, and let $P(\mathcal{M}_t \mid \mathcal{D})$ denote the probability $P(X = \mathcal{M}_t \mid \mathcal{D})$. In our simulated annealing process, we generate a new candidate $\mathcal{M}_{t+1}$ for the next value of X by changing the cluster indices $c(h)$ of a randomly chosen small subset of data vectors $\vec{d}_h$. The resulting new partition vector $\mathcal{M}_{t+1}$ is accepted as the next value for the random variable X with probability

$$p_t(\mathcal{M}_{t+1}) = \frac{1}{1 + (\frac{P(\mathcal{M}_t \mid \mathcal{D})}{P(\mathcal{M}_{t+1} \mid \mathcal{D})})^{\frac{1}{T(t)}}}, \tag{8}$$

where $T(t)$ is a monotonicly decreasing function converging to zero as t approaches infinity, and the conditional probabilities $P(\mathcal{M}_t \mid \mathcal{D})$ and $P(\mathcal{M}_{t+1} \mid \mathcal{D})$ are computed using equation (6). The value of the function T is usually called the (computational) *temperature* of the process. Without the temperature parameter, the probability p_t is identical to the acceptance probability proposed by Barker in [2]. An alternative form for the acceptance probability, proposed originally by Metropolis et al. in [24], is given by

$$p_t(\mathcal{M}_{t+1}) = \begin{cases} 1 & , \text{ if } \frac{P(\mathcal{M}_{t+1} \mid \mathcal{D})}{P(\mathcal{M}_t \mid \mathcal{D})} \geq 1, \\ (\frac{P(\mathcal{M}_{t+1} \mid \mathcal{D})}{P(\mathcal{M}_t \mid \mathcal{D})})^{\frac{1}{T(t)}} & , \text{ if } \frac{P(\mathcal{M}_{t+1} \mid \mathcal{D})}{P(\mathcal{M}_t \mid \mathcal{D})} < 1. \end{cases} \tag{9}$$

It has been argued that in practice the Metropolis form should be preferred since in the case of equally probable states, Barker's method changes the state with probability 1/2, whereas Metropolis' method changes the state with probability 1, thus offering possibly a better sampling of the states. However, it should be noted that the stochastic process generated by using the Barker's formula can be implemented extremely efficiently by using a massively parallel architecture [26, 27]. Theoretically there is no preference for either of these models, since using the candidate generation scheme described above both methods lead to a stochastic process fulfilling the condition (7) (see [26]).

Algorithm 1 (Searching models by simulated annealing)

1. /* start with a random clustering: */
   for h = 1 to N do
   (a) $c_0(h)$ = RANDINT(1,N);
2. Construct a prototype tree model $\mathcal{M}_0$ corresponding to the initial clustering $\vec{c}_0$;
3. for t = 0 to ∞ do
   (a) Compute the new computational temperature $T(t)$;
   (b) If $T(t) = 0$ then stop;
   (c) /* generate a new clustering by changing randomly a fraction of the cluster indices */
   for h = 0 to N do
      i. If (RANDREAL(0,1) $< \theta$)
      then $c_{t+1}(h)$ = RANDINT(1,N); /* choose randomly a new cluster */
      else $c_{t+1}(h) = c_t(h)$;
   (d) Construct a new prototype tree model $\mathcal{M}_{t+1}$ corresponding to the new clustering $\vec{c}_{t+1}$;
   (e) Compute the probability $p_t(\mathcal{M}_{t+1})$ by using equation (8) or (9);
   (f) If ($p_t(\mathcal{M}_{t+1})$ < RANDREAL(0,1))
   then $\vec{c}_{t+1} = \vec{c}_t$; /* reject new clustering, restore old value */

The parameter θ controls the number of data vectors to be moved from their current cluster to a new, randomly chosen cluster. Theoretically the value of θ can be anything between 0 and 1, but in our experiments we have typically used small values between $1/N$ and 0.05.

If the computational temperature $T(t)$ is decreased slowly enough, the simulated annealing algorithm given will converge to the maximum probability clustering resulting in maximum probability model $\mathcal{M}_*$ [1]. Unfortunately, finding a proper cooling schedule for decreasing the temperature may be difficult in practice [26].

6 Experiments

To test the proposed framework in practice, we ran a series of experiments by using a set of medical data dealing with diagnosis of *Nephropathia epidemica (NE)*, which is the mildest form of hemorrhagic fevers with renal syndrome [10]. In these experiments, each data vector consisted of values of 33 attributes, the last of which gives the diagnosis of the patient (NE or non-NE). The training set consisted of 210 vectors with 70 positive (NE) and 140 negative (non-NE) cases, and the testing set contained 105 vectors, of which 35 were positive and 70 negative. The diagnoses of the data vectors were obtained by complex viral antibody tests, which can not normally be used for diagnosing the patients in hospitals. In the following, by *success rate* we mean the proportional number of correct classifications in the testing set, after the model had been trained with the training set.

For comparison, the data was first given to a group of 15 students attending a neural network course, and each student was permitted to use whatever model they chose to experiment with the data. The resulting success rates ranged from 76.2% to 91.5%. The two clear winners with the success rates of 91.5% used Backpropagation (BP) (see e.g. [34]) and Learning Vector Quantization (LVQ) [17], respectively. The best BP result was obtained by a 32-5-3-2 network, and by adding 20% noise to the training data. However, finding this setup required a lot of experiments, and constant manual tuning of the parameters of the algorithm during the training. The best LVQ result was obtained by first using a statistical analysis package for finding a suitable number of reference vectors, and then learning the vectors by using the algorithm versions LVQ1, LVQ2 and

LVQ3, in that order. At each stage, the training was repeated 50–150 times for searching for the optimal setting of the parameters.

For the simulated annealing approach described in the previous section, we decided at this initial phase not to use the probability $P(\mathcal{M})$ in computing the evaluation criterion, since the search space with over 210^{210} possible vectors requires very long annealing schedules, and hence much computing resources, which at this time were not available. In addition, it was not yet clear to us what is in practice the optimal value r, the number of bits to represent a truncated real value, which is needed for computing $S(\mathcal{M})$ in (5). It seems that too large values of r give too high a penalty to models with many classes, whereas training with too small values for r results in too complex models. Consequently, we chose to try the simulated annealing approach first with a constant number of classes (in which case $P(\mathcal{M})$ is constant and can be ignored), starting with two classes, and repeating the tests with increasing number of classes. To our surprise, this approach produced a success rate of 92.4% already with only two clusters. We are currently still continuing with our experiments, and based on these very restricted early tests we do not wish to make any claims of the superiority of the Bayesian approach against the neural network techniques. Nevertheless, it looks evident that with the Bayesian prototype tree learning approach it is easy to find simple models with a performance comparable to that of much more complex neural network architectures, with the additional benefit that the resulting models are also semantically interpretable, as the class prototype vectors represent probability distributions inside clusters of similar data vectors.

7 Conclusion

We have presented a method for deriving probabilistic Bayesian network models from data. In this work, we have restricted ourselves to a class of simple tree-structured Bayesian prototype tree models, although in principle the approach presented can be adapted also for learning multi-connected Bayesian networks. The reasons for choosing the family of Bayesian prototype tree models was twofold: firstly, the resulting Bayesian network models can be implemented extremely efficiently, especially if massively parallel hardware is available. Secondly, the prototype tree structure has an intuitively appealing, clear semantic interpretation as a memory-based reasoning system. The learning algorithm introduced is based on a stochastic simulated annealing search algorithm. We are currently experimenting with the proposed simulated annealing learning algorithm for Bayesian prototype trees, and in the future we plan to apply the algorithm to well-known benchmark problems in order to compare our approach with alternative methods.

References

[1] Aarts, E., and Korst, J., *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, Chichester 1989.

[2] Barker, A.A., Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Aust. J. Phys.* 18 (1965), 119–133.

[3] Brown, D.E., and Huntley, C.L., A practical application of simulated annealing to clustering. *Pattern Recognition* 25 (1992), 401–412.

[4] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D., AutoClass: a Bayesian classification system. Pp. 54–64 in *Proc. of the Fifth International Conference on Machine Learning*, Ann Arbor, June 1988 (Morgan Kaufmann).

[5] Cheeseman, P., On finding the most probable model. Pp. 73–95 in J. Shrager and P. Langley (eds.), *Computational Models of Scientific Discovery and Theory Formation.* Morgan Kaufmann, 1990.

[6] Cooper, G.F., The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42 (1990), 393–405.

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

[8] Csiszár, I. and Tusnády, G., Information geometry and alternating minimization procedure. *Statistics & Decisions*, Supplement Issue 1 (1984), 205–237.

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

[10] Forsström, J. Machine learning in clinical medicine by knowledge acquisition from patient databases. Ph.D. Thesis, Departments of Medicine and Clinical Chemistry, University of Turku. Annales Universitatis Turkuensis, Ser. D, Report 92. University of Turku, 1992.

[11] Geman, S. and Geman, D., Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 6 (1984), 721–741.

[12] Gyllenberg, M., Gyllenberg, H.G., Koski, T., and Schindler, J., Non-uniqueness of numerical taxonomic structures. *Binary* 5 (1993), 138–144.

[13] Hsu, W., Hsu, L.S., and Tenorio, M.F., The Clusnet algorithm and time series prediction. *International Journal of Neural Systems* 4 (1993), 247–255.

[14] Kirkpatrick, S., Gelatt, D. and Vecchi, M.P., Optimization by simulated annealing. *Science 220* (1983), 671–680.

[15] Kitano, H., Challenges of massive parallelism. Pp. 813–834 in *Proc. of IJCAI-93, the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, August 1993 (Morgan Kaufmann).

[16] Klein, R.W. and Dubes, R.C. Experiments in projection and clustering by simulated annealing. *Pattern Recognition* 22 (1989), 213–220.

[17] Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin 1988.

[18] Kononenko, I., Successive naive Bayesian classifier. *Informatica* 17 (1993) , 167–174.

[19] Langley, P., Induction of recursive Bayesian classifiers. Pp. 153–164 in P.B. Brazdil (ed.), *Proc. of ECML-93, European Conference on Machine Learning*, Vienna, Austria, April 1993 (Springer-Verlag).

[20] Lam W., and Bacchus, F., Using causal information and local measures to learn Bayesian networks. Pp. 243–250 in D. Heckerman and A. Mamdani (eds.), *Proc. of the Ninth Conference on Uncertainty in Artificial Intelligence*, Washington, D.C., July 1993 (Morgan Kaufmann).

[21] Lee S., and Shimoji, S., BAYESNET: Bayesian classification network based on biased random competition using Gaussian kernels. Pp. 1354–1359 in *Proc. of the IEEE International Conf. on Neural Networks*, San Francisco, March 1993 (IEEE Press).

[22] MacQueen, J. Some methods of classification and analysis of multivariate observation. Pp. 281–297 in L.M.LeCam and J.Neyman (eds.), *Proc. 5th Berkeley Symposium on Math. Stat., and Prob.*, 1967.

[23] Marroquin, J.L. and Girosi, F. Some extensions of the k-means algorithm for image segmentation and pattern classification. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, A.I. Memo 1390, C.B.C.L. Paper No. 079, MIT January 1993.

[24] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, M.N. and Teller, E., Equations of state calculations by fast computing machines. *Journal of Chem. Phys.* 21 (1953), 1087–1092.

[25] Moody, J., and Darken, C., Fast learning in networks of locally-tuned processing units. *Neural Computation* 1 (1989), 281–294.

[26] Myllymäki, P., Bayesian reasoning by stochastic neural networks. Ph. Lic. Thesis, Report C-1993-67, Department of Computer Science, University of Helsinki, December 1993.

[27] Myllymäki, P., Using Bayesian networks for incorporating probabilistic a priori knowledge into Boltzmann machines. Pp. 97–102 in *Proc. of SOUTHCON'94*, Orlando, March 1994 (IEEE Press).

[28] Myllymäki, P., and Tirri, H., Bayesian case-based reasoning with neural networks. Pp. 422–427 in *Proc. of the IEEE International Conf. on Neural Networks*, San Francisco, March 1993 (IEEE Press).

[29] Myllymäki, P., and Tirri, H., Massively parallel case-based reasoning with probabilistic similarity metrics. Pp. 48–53 in M.M. Richter, S. Wess, K.-D. Althoff and F. Maurer (eds.), *Proc. of the First European Workshop on Case-Based Reasoning*, University of Kaiserslautern, November 1993. SEKI Report SR-93-12 (SFB 314), University of Kaiserslautern.

[30] Myllymäki, P., and Tirri, H., Learning in neural networks with Bayesian prototypes. Pp. 60–64 in *Proc. of SOUTHCON'94*, Orlando, March 1994 (IEEE Press).

[31] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[32] Rissanen, J., *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[33] Robinson, W.R., Counting unlabeled acyclic digraphs. In C.H.C. Little (ed.), *Lecture notes in mathematics 622: Combinatorial mathematics.* Springer-Verlag, 1977.

[34] Rumelhart, D.E., Hinton G.E. and Williams R.J., Learning internal representations by error propagation. Pp. 318–362 in D.E.Rumelhart and J.L.McClelland (eds.), *Parallel Distributed Processing.* MIT Press, 1986.

[35] Selim, S.Z., and Alsultan, K., A simulated annealing algorithm for the clustering problem. *Pattern Recognition* 24 (1991) 10, 1003–1008.

[36] Selim, S.Z., and Ismail, M.A. K-Means-type algorithms: a generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984) 1, 81–87.

[37] Specht, D.F., Probabilistic neural networks. *Neural Networks* 3, 109–118, 1990.