168 UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000

## Mix-nets: Factored Mixtures of Gaussians in Bayesian Networks with Mixed Continuous And Discrete Variables

Abstract

Scott Davies and Andrew Moore

School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213 [scottd, awm]@cs.cmu.edu

P(X-) as P(X-) = Ni

-

where N is the num-



Recently developed techniques have made it pos­ sible to quickly learn accurate probability density functions from data in low-dimensional continu­ ous spaces. In particular, mixtures of Gaussians can be fitted to data very quickly using an ac­ celerated EM algorithm that employs multires­ olution kd-trees (Moore, 1999). In this paper, we propose a kind of Bayesian network in which low-dimensional mixtures of Gaussians over dif­ ferent subsets of the domain's variables are com­ bined into a coherentjoint probability model over the entire domain. The network is also capable of modeling complex dependencies between dis­ crete variables and continuous variables without requiring discretization of the continuous vari­ ables. We present efficient heuristic algorithms for automatically learning these networks from data, and perform comparative experiments illus­ trating how well these networks model real sci­ entific data and synthetic data. We also briefly discuss some possible improvements to the net­ works, as well as possible applications.

1. INTRODUCTION

Bayesian networks are a popular method for represent­ ing joint probability distributions over many variables. A

Bayesian network contains a directed acyclic graph G with one vertex Vi in the graph for each variable Xi in the do­ main. The directed edges in the graph specify a set of inde­ pendence relationships between the variables. Define rfi to be the set of variables whose nodes in the graph are "par­

ents" of v;. The set of independence relationships spec­ ified by G is then as follows: given the values of rfi but no other information, xi is conditionally independent of

all variables corresponding to nodes that are not Vi's de­ scendants in the graph. These independence relationships allows us to decompose the joint probability distribution

## f1 =l P(Xiiiii),

ber of variables in the domain. Thus, if in addition to G we also specify P(Xilrli) for every variable Xi, then we have specified a valid probability distribution P(X) over the entire domain.

Bayesian networks are most commonly used in situations where all the variables are discrete; if continuous variables are modeled at all, they are typically assumed to follow simple parametric distributions such as Gaussians. Some researchers have recently investigated the use of complex continuous distributions within Bayesian networks; for ex­ ample, weighted sums of Gaussians have been used to ap­ proximate conditional probability density functions (Driver and Morrell, 1995). Such complex distributions over con­ tinuous variables are usually quite computationally expen­ sive to learn. This expense may not be too problematic if an appropriate Bayesian network structure is known be­ forehand. On the other hand, if the dependencies between variables are not known a *priori* and the structure must be learned from data, then the number of conditional distri­ butions that must be learned and tested while a structure­ learning algorithm searches for a good network can become unmanageably large.

However, very fast algorithms for learning complex joint probability densities over small sets of continuous variables have recently been developed (Moore, 1999). This pa­ per investigates how these algorithms can be used to learn Bayesian networks over many variables, each of which can be either continuous or discrete.

1. MIX-NETS

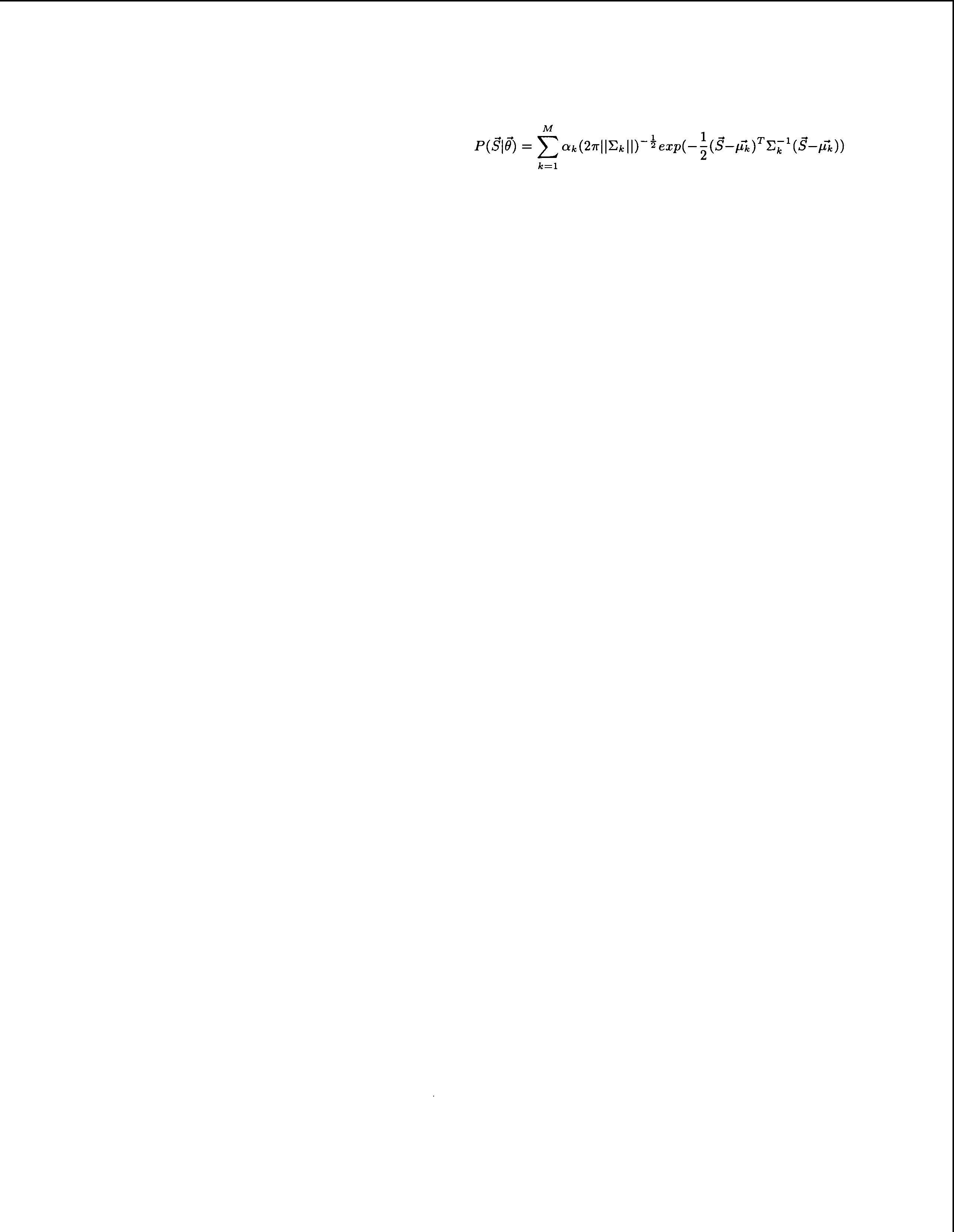
Suppose we have a very fast, black-box algorithm A geared not towards finding accurate conditional models of the form

Pi(Xilrli), but rather towards finding accurate *joint* prob­

ability models Pi(§;) over subsets of variables *5;,* such as Pi(Xi, rfi). Furthermore, suppose it is only capable of

generating joint models for relatively small subsets of the variables, and that the models returned for different sub-

UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000 169



sets of variables are not necessarily consistent. Can we still combine many different models generated by A into a valid probability distribution over the entire space?

Fortunately, the answer is yes, as long as the models re­ turned by A can be marginalized exactly. If for any given

Pi(Xi,Ifi) we can compute a marginal distribution Pi(ITi) that is consistent with it, the!! we can empl?,Y Pi as� con­ ditional distribution Pi(Xiiiii) = Pi(Xi,IIi)/Pi(IIi). In

this case, given a directed acyclic graph G specifying a

Bayesian network structure over N variables, we can sim­ ply use A to acquire N models Pi(Xi,Ifi), marginalize these models, and string them together to form a distribu­

tion over the entire space:

N

## P(X) = II (Pi(Xi,Ifi)/Pi(ITi))

i=l

Even though the marginals of different Pi's may be incon­ sistent with each other, the Pi's are only *used* conditionally, and in a manner that prevents these inconsistencies from actually causing the overall model to become inconsistent. Of course, the fact that there are inconsistencies at all - suppressed or not - means that there is a certain amount of redundancy in the overall model. However, if allowing such redundancy lets us employ a particularly fast and ef­ fective model-learning algorithm A, it may be worth it.

Joint models over subsets of variables have been similarly conditionalized in previous research in order to use them within Bayesian networks. For example, the conditional distribution of each variable in the network given its par­ ents can be modeled by conditionalizing another "embed­ ded" Bayesian network that specifies the joint between the variable and its parents (Heckerman and Meek, 1997a). (Some theoretical issues concerning the interdependence of parameters in such models appear in (Heckerman and Meek, 1997a) and (Heckerman and Meek, 1997b).) Joint distributions formed by convolving a Gaussian kernel func­ tion with each of the datapoints have also been condition­ alized for use in Bayesian networks over continuous vari­ ables (Hofmann and Tresp, 1995).

* 1. HANDLING CONTINUOUS VARIABLES

Suppose for the moment that X contains only continuous values. What sorts of models might we want A to return?

One powerful type of model for representing probability density functions over small sets of variables is a *Gaussian mixture model* (see e.g. (Duda and Hart, 1973)). Let sj

represent the values that the *jth* datapoint in the dataset D

assigns to a variable set of interest §. In a Gaussian mix­

ture model over S, we assume that the data are generated independently through the following process: for each sj in tum, nature begins by randomly picking a class, Ck. from a discrete set of classes c1, ... , CM. Then nature draws sj

from a multidimensional Gaussian whose mean vector 1-'·k and covariance matrix I:k depend on the class. This pro-

duces a distribution of the following mathematical form:

where ak represents the probability of a point coming from the *kth* class, and if denotes the entire set of the mixture's

parameters (the a's, f.L's and I:'s). It is possible to model any continuous probability distribution with arbitrary accu­ racy by using a mixture with a sufficiently large M.

Given a Gaussian mixture model Pi(Xi,Ifi), it is easy to compute the marginalization Pi(ITi): the marginal mixture

has the same number of Gaussians as the original mix­ ture, with the same a's. The means and covariances of the marginal mixture are simply the means and covariances of the original mixture with all elements corresponding to the

variable Xi removed. Thus, Gaussian mixture models are

suitable for combining into global joint probability density

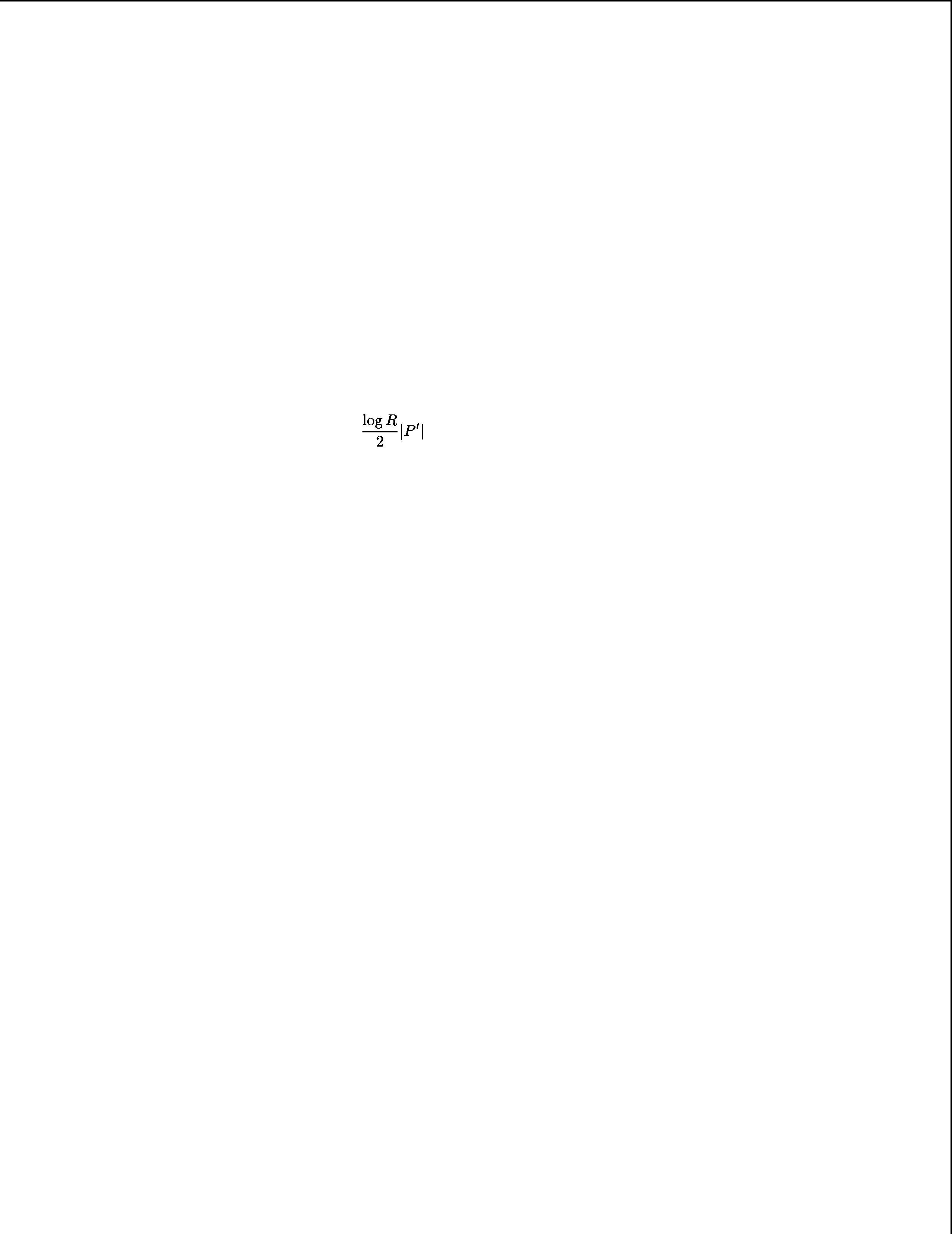
functions using the methodology described in section 2, as­ suming all variables in the domain are continuous. This is the class of models we employ for continuous variables in this paper, although many other classes may be used in an analogous fashion.

The functional form of the conditional distribution we use is similar to that employed in previous research by con­ ditionalizing a joint distribution formed by convolving a Gaussian kernel function with all the datapoints (Hofmann and Tresp, 1995). The differences are that our distribu­ tions use fewer Gaussians, but these Gaussians have vary­ ing weights and varying non-diagonal covariance matrices. The use of fewer Gaussians makes our method more suit­ able for some applications such as compression, and may speed up inference. Our method may also yield more ac­ curate models in many situations, but we have yet to verify this experimentally.

* + 1. Learning Gaussian Mixtures From Data

The EM algorithm is a popular method for learning mixture models from data (see, e.g., (Dempster et a!., 1977)). The algorithm is an iterative algorithm with two steps per itera­ tion. The Expectation or "E" step calculates the distribution over the unobserved mixture component variables, using the current estimates for the model's parameters. The Max­ imization or "M" step then re-estimates the model's param­ eters to maximize the likelihood of both the observed data and the unobserved mixture component variables, assum­ ing the distribution over mixture components calculated in the previous "E" step is correct. Each iteration of the EM algorithm increases the likelihood of the observed data or leaves it unchanged; if it leaves it unchanged, this usually indicates that the likelihood is at a local maximum. Un­ fortunately, each iteration of the basic algorithm described above is slow, since it requires a entire pass through the dataset. Instead, we use an accelerated EM algorithm in

170 UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000



which multiresolution kd-trees (Moore et al., 1997) are used to dramatically reduce the computational cost of each iteration (Moore, 1999). We refer the interested reader to this previous paper (Moore, 1999) for details.

An important remaining issue is how to choose the appro­ priate number of Gaussians, M, for the mixture. If we re­ strict ourselves to too few Gaussians, we will fail to model the data accurately; on the other hand, if we allow ourselves too many, then we may "overfit" the data and our model may generalize poorly. A popular way of dealing with this tradeoff is to choose the model maximizing a scoring func­ tion that includes penalty terms related to the number of parameters in the model. We employ the Bayesian Infor­ mation Criterion (Schwarz, 1978), or BIC, to choose be­

tween mixtures with different numbers of Gaussians. The BIC score for a probability model P' (S) is as follows:

# BIC(P') = logP'(Ds)-

where Ds is the dataset D restricted to the variables of interest S, R is the number of datapoints in the dataset, and

IP'I is the number of parameters in P'.

Rather than re-running the EM algorithm to convergence for many different choices of M and choosing the resulting mixture that maximizes the BIC score, we use a heuristic algorithm that starts with a small number of Gaussians and stochastically tries adding or deleting Gaussians as the EM algorithm progresses. The details of this algorithm are de­ scribed in a separate paper (Sand and Moore, 2000).

* 1. HANDLING DISCRETE VARIABLES

Suppose now that a set of variables *�* we wish to model includes discrete variables as well as continuous variables.

Let Qi be the discrete variables in *�.* and Ci the contin­ uous variables in *�* - One simple model for Pi(Qi,Ci) is a lookup table with an entry for each possible set qi of as­ signments to Qi. The entry in the table corresponding to a

particular rji contains two things: the marginal probability Pi(qi), and a Gaussian mixture modeling the conditional distribution pi(ciIri). Let us refer to tables of this form as *mixture tables.* We obtain the mixture table's estimate for each Pi(qi) directly from the data: it is simply the fraction of the records in the dataset that assigns the values qi to Qi.

Given an algorithm A for learning Gaussian mixtures from

continuous data, we use it to generate each conditional dis­ tribution Pi(Cilqi) in the mixture table by calling it with

the subset of the dataset D that is consistent with the val­ ues specified by qi.

Suppose now that we are given a network structure over the entire set of variables, and for each variable Xi we are given a mixture table for Pi(*�*) = Pi(Xi,:rfi). We must

now calculate new mixture tables for the marginal distri­ butions Pi(:rfi) so that we can use them for the conditional

distributions pi(XiiTii) = pi(Xi,:rfi)Ipi(Tii). Let ci rep­ resent the continuous variables in {Xi} U :rfi; Qi represent the discrete variables in {xi} u :rfi; c�. represent the con­ tinuous variables in :rfi; and Q7I, represent the discrete vari­ ables in :rfi. (Either Q7I, = Qi or c�. = Ci, depending on whether xi is continuous or discrete.)

If Xi is continuous, then the marginalized mixture table for Pi(:rfi) has the same number of entries as the original table for Pi(Xi,:rfi), and the estimates for P(Qi) in the

marginalized table are the same as in the original table. For each combination of assignments to Qi, we marginalize the appropriate Gaussian mixture Pi(CiiQi) = Pi(CdQ7IJ in the original table to a new mixture Pi(C�, IQ�, ), and use

this new mixture in the corresponding spot in the marginal­ ized table.

If Xi is discrete, then for each combination of values for Q7I, we combine several different Gaussian mixtures for various Pi(C�, IQi)'s into a new Gaussian mixture for Pi(C�, IQ7IJ. First, the values of Pi(Q�.) in the

marginalized table are computed from the original table as Pi(Q7IJ = Lx, Pi(Xi,Q7IJ. Pi(XiiQ�.) is then calcu­ lated as Pi(Xi,Q7IJ/Pi(Q7IJ. Finally, we combine the Gaussian mixtures corresponding to different values of Xi

according to the relationship

# Pi(c�. IQ7IJ = L Pi(XiiQ7IJPi(c�.IQi)·

## x,

We have now described the steps necessary to use mixture tables in order to parameterize Bayesian networks over do­ mains with both discrete and continuous variables. Note that mixture tables are not particularly well-suited for deal­ ing with discrete variables that can take on many possible values, or for scenarios involving many dependent discrete variables - in such situations, the continuous data will be shattered into many separate Gaussian mixtures, each of which will have little support. Better ways of dealing with discrete variables are undoubtedly possible, but we leave them for future research (see section 6.3).

1. LEARNING MIX-NET STRUCTURES

Given a Bayesian network structure over a domain with both discrete and continuous variables, we now know how to learn mixture tables describing the joint probability of each variable and its parent variables, and how to marginal­ ize these mixture tables to obtain the conditional distribu­ tions needed to compute a coherent probability function over the entire domain. But what if we don't know *a pri­ ori* what dependencies exist between the variables in the domain - can we learn these dependencies automatically and find the best Bayesian network structure on our own, or at least find a "good" network structure?

UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000 171



In general, finding the optimal Bayesian network struc­ ture with which to model a given dataset is NP­ complete (Chickering, 1996), even when all the data is discrete and there are no missing values or hidden vari­ ables. A popular heuristic approach to finding networks that model discrete data well is to hillclimb over network structures, using a scoring function such as the BIC as the criterion to maximize. (See, e.g., (Heckerman et a!., 1995).) Unfortunately, hillclimbing usually requires scor­ ing a very large number of networks. W hile our algorithm for learning Gaussian mixtures from data is comparatively fast for the complex task it performs, it is still too expen­ sive to re-run on the hundreds of thousands of different variable subsets that would be necessary to parameterize all the networks tested over an extensive hillclimbing run. (Such a hillclimbing algorithm has previously been used to find Bayesian networks suitable for modeling continu­ ous data with complex distributions (Hofmann and Tresp, 1995), but in practice this method is restricted to datasets with relatively small numbers of variables and datapoints.)

However, there are other heuristic algorithms that often find networks very close in quality to those found by hillclimb­ ing but with much less computation. A frequently used class of algorithms involves measuring all pairwise inter­ actions between the variables, and then constructing a net­ work that models the strongest of these pairwise interac­ tions (e.g. (Chow and Liu, 1968; Sahami, 1996; Friedman et a!., 1999)). We use such an algorithm in this paper to au­ tomatically learn the structures of our Bayesian networks.

In order to measure the pairwise interactions between the variables, we start with an empty Bayesian network B, in which there no arcs - i.e., in which all variables are as­ sumed to be independent. We use our mixture-learning al­ gorithm to calculate B,'s parameters, and then calculate B,'s BIC score. (The number of parameters in the network is computed as the sum of the parameters in each network node, where the parameters of a node for variable X;are

the parameters of P(;X;, II).) We then calculate the BIC

score of every possible Bayesian network containing ex­ actly one arc. With N variables, there are O(N2) such networks. Let B;j denote the network with a single arc from X;to Xi. Note that to compute the BIC score of B;j, we need not recompute the mixture tables for any variable

other than Xj, since the others can simply be copied from B,. Now, define I(X;, Xi), the "importance" of the depen­ dency between variable X;and Xj, as follows:

I(X;, Xi)= BIC(B;j)- BIC(B,).

After computing all the I(X;, Xj)'s, we initialize our cur­ rent working network Bto the empty network B., and then greedily add arcs to Busing the I(X;, Xj)'s as hints for what arcs to try adding next. At any given point in the algorithm, the set of variables is split into two mutually ex­ clusive subsets, *DONE* and *PENDING.* All variables begin

* B := B" PENDING:= the set of all variables, DONE:={}
* While there are still variables-in PENDING:
  + Consider all pairs of variables Xd and Xp such that Xd is in DONE and Xp is in PENDING. Of these, let X:Ja• and x;a• be the pair of variables that maximizes I (Xd, XP ) . Our algorithm selects

x;a• as the next variable to consider adding arcs to. (Ties are han­

dled arbitrarily, as is the case where DONE is currently empty.)

* + Let K' = min(K, I DONEl). where K is a user-defined pa­ rameter. Let X�, XJ, ...X:f' denote the K' variables in DONE with the highest values of !(X�, X;:'a'), in descending order of

## !(X�, x;;a• ).

* + Fori = 1 to K':
    - If x;;a• now has MAX.PARS parents in B, or

if I(X�, X;:'a. ) is less than zero, break out of the for loop overi and do not consider adding any more parents to x;a•.

* + - Let B' be a network identical to B except with an additional arc from X� to x;;a•. Call our mixture-learning algorithm to update the parameters for X;'"' 's node in B', and compute

BIC(B').

* + - IfBIC(B') > BIC(B),B := B'.
  + Move X;'"' from PENDING to DONE.

Figure 1: Our network structure learning algorithm.

in the *PENDING* set. Our algorithm proceeds by selecting a variable in the *PENDING* set, adding arcs to that variable from other variables in the *DONE* set, moving the variable to the *DONE* set, and repeating until all variables are in *DONE.* High-level pseudo-code for the algorithm appears in Figure 1.

The algorithm generates O(N2) mixture tables contain­ ing two variables each in order to calculate all the pair­ wise dependency strengths I(X;, Xj), and then O(N \* K) more tables containing MAXPARS + 1 or fewer vari­

ables each during the greedy network construction. K is a

user-defined parameter that determines the maximum num­ ber of potential parents evaluated for each variable during the greedy network construction.

If *MAXPARS* is set to 1 and I(X;, Xj) is symmetric, then our heuristic algorithm reduces to a maximum spanning tree algorithm (or to a maximum-weight forest algorithm if some of the I's are negative). Out of all possible Bayesian networks in which each variable has at most one parent, this

maximum spanning tree is the Bayesian network *B�pt* that

maximizes the scoring function. (This is a trivial general­ ization of the well-known algorithm (Chow and Liu, 1968) for the case where the unpenalized log-likelihood is the ob­ jective criteria being maximized.) If *MAXPARS* is set above

1, our heuristic algorithm will always model a superset of the dependencies in *B�pt•* and will always find a network with at least as high a BIG score as *B�pt*'s.

There are a few details that prevent our I(X;, Xi)'s from being perfectly symmetric. (See (Davies and Moore, 2000).) However, I is close enough to symmetric that it's often worth simply assuming that it is symmetric, since

172 UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000

this cuts down the number of calls we need to make to our mixture-learning algorithm in order to calculate the I(X;, Xj)'s by roughly a factor of 2.

Since learning joint distributions involving real variables is expensive, calling our mixture table generator even just O(N2) times to measure all of the I(Xi,Xj)'s can take a

prohibitive amount of time. We note that the I(X;, Xj)'s are only used to choose the order in which the algorithm selects variables to move from *PENDING* to *DONE,* and to select which arcs to try adding to the graph. The ac­ tual values of I(X;,Xj) are irrelevant - the only things that matter are their ranks and whether they are greater than zero. Thus, in order to reduce the expense of com­ puting the I(X;,Xj)'s, we can try computing them on a *discretized* version of the dataset rather than the original dataset that includes continuous values. The resulting ranks of I(Xi , Xj) will not generally be the same as they would if they were computed from the original dataset, but we would expect them to be highly correlated in many practi­ cal circumstances.

Our structure-learning algorithm is similar to the "Limited Dependence Bayesian Classifiers" previously employed to learn networks for classification (Saharni, 1996), except that our networks have no special target variable, and we add the potential parents to a given node one at a time to ensure that each actually increases the network's score. Al­ ternatively, our learning algorithm can be viewed as a re­ striction of the "Sparse Candidate" algorithm (Friedman et al., 1999) in which only one set of candidate parents is generated for each node, and in which the search over network structures restricted to those candidate parents is performed greedily. (We have also previously used a very similar algorithm for learningnetworks with which to com­ press discrete datasets (Davies and Moore, 1999).)

1. EXPERIMENTS
   1. ALGORITHMS

We compare the performance of the network-learning al­ gorithm described above to the performance of four other algorithms, each of which is designed to be similar to our network-learning algorithm except in one important respect. First we describe a few details about how our primary network-learning algorithm is used in our experi­ ments, and then we describe the four alternative algorithms.

The Mix-Net algorithm is our primary network-learning algorithm. For our experiments on both datasets, we set *MAXPARS* to 3 and K to 6. When generating any given Gaussian mixture, we give our accelerated EM algorithm thirty seconds to find the best mixture it can. In order to make the most of these thirty-second intervals, we also

limit our overall training algorithm to using a sample of at most I 0,000 datapoints from the training set. Rather than

computing the I(Xi, Xj)'s with the original dataset, we compute them with a version of the dataset in which each continuous variable has been discretized to 16 different val­ ues. The boundaries of the 16 bins for each variable's dis­ cretization are chosen so that the numbers of datapoints in each bin are approximately equal.

Mixture tables containing many discrete variables (or a few discrete variables each of which can take on many values) can severely overfit data, since some combinations of the discrete variables may occur rarely in the data. For now, we attempt to address this problem by slightly smoothing the distributions represented in the tables. (See (Davies and Moore, 2000) for details.)

The Independent Mixtures algorithm is identical to our mix-net learning algorithm in almost all respects; the main difference is that here the *MAXPARS* parameter has been set to zero, thus forcing all variables to be modeled inde­ pendently. We also give this algorithm more time to learn each individual Gaussian mixture, so that it is given a total amount of computational time at least as great as that used by our mix-net learning algorithm.

The Trees algorithm is identical to our mix-net algorithm in all respects except that the *MAXPARS* parameter has been set to one, and we give it more time to learn each individual Gaussian mixture (as we did for the Independent Mixtures algorithm).

The Single-Gaussian algorithm is identical to our pri­ mary network-learning algorithm except for the follow­

ing differences. W hen learning a given Gaussian mixture Pi(*�* IQ i), we use only a single multidimensional Gaus­

sian. (Note, however, that some of the marginal distribu­ tions Pi(C�; IQ*�*.) may contain multiple Gaussians when

the variable marginalized away is discrete.) Since single Gaussians are much easier to learn in high-dimensional spaces than mixtures are, we allow this single-Gaussian al­ gorithm much more freedom in creating large mixtures. We

set both *MAXPARS* and K to the total number of variables

in the domain minus one. We also allow the algorithm to use all datapoints in the training set rather than restrict it to a sample of 10,000. Finally, we use the original real-valued dataset rather than a discretized version of the dataset when computing the I(Xi,Xj)'s.

The Pseudo-Discrete algorithm is similar to our primary network-learning algorithm in that it uses the same sort of greedy algorithm to select which arcs to try adding to the network. However, the networks this algorithm pro­ duces do not employ Gaussian mixtures. Instead, the dis­ tributions it uses are closely related to the distributions that would be modeled by a Bayesian network for a completely discretized version of the dataset. For each continuous vari­ able X;in the domain, we break X'; s range into F buckets. The boundaries of the buckets are chosen so that the num­ ber of datapoints lying within each bucket is approximately

UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000 173



equal. The conditional distribution for Xi is modeled with a table containing one entry for every combination of its

parent variables, where each continuous parent variable's value is discretized according to the F buckets we have se­

lected for that parent variable. Each entry in the table con­ tains a histogram for xi recording the conditional proba­ bility that xi's value lies within the boundaries of each of Xi's F buckets. We then translate the conditional probabil­

ity associated with each bucket into a conditional probabil­ ity density spread uniformly throughout the range of that bucket. (Discrete variables are handled in a similar man­ ner, except the translation from conditional probabilities to conditional probability densities is not performed.)

When performing experiments with the Pseudo-Discrete network-learning algorithm, we re-run it for several differ­ ent choices of F: 2, 4, 8, 16, 32, and 64. Of the resulting

networks, we pick the one that maximizes the BIC. When the algorithm uses a particular value for F, the I(Xi,Xj ) ' s are computed using a version of the dataset that has been

discretized accordingly, and then arcs are added greedily as in our mix-net learning algorithm. The networks produced by this algorithm do not have redundant parameters as our mix-nets do, as each node contains only a model of its vari­ able's conditional distribution given its parents rather than a joint distribution.

Note that there are better ways of discretizing real vari­ ables in Bayesian networks (e.g. (Kozlov and Koller, 1997; Monti and Cooper, 1998a)); the simple discretization al­ gorithm discussed here and currently implemented for our experiments is certainly not state-of-the-art.

* 1. DATASE TS AND RESULTS

We tested the previously described algorithms on two dif­ ferent datasets taken from real scientific experiments. The "Bio" dataset contains data from a high-throughput biolog­ ical cell assay. There are 12,671 records and 31 variables. 26 of the variables are continuous; the other five are dis­ crete. Each discrete variable can take on either two or three different possible values. The "Astro" dataset contains data taken from the Sloan Digital Sky Survey, an extensive as­ tronomical survey currently in progress. This dataset con­ tains 111,456 records and 68 variables. 65 of the variables are continuous; the other three are discrete, with arities ranging from three to 81. All continuous values are scaled

to lie within [0 , I] after a small amount of uniform noise

is added to them to prevent potential problems caused by point distributions or deterministic relationships in the data.

For each dataset and each algorithm, we performed ten­ fold cross-validation, and recorded the log-likelihoods of the test sets given the resulting models. Figure 2 shows the mean log-likelihoods of the test sets according to models generated by our five network-learning algorithms, as well as the standard deviation of the means. (Note that the log-

likelihoods are positive since most of the variables are con­ tinuous and bounded within [0 , 1], which implies that the models usually assign probability densities greater than one to regions of the space containing most of the datapoints. The probability distributions modeled by the networks are properly normalized, however.)

On the Bio dataset, our primary mix-net learner achieved significantly higher log-likelihood scores than the other four model learners. The fact that it significantly outper­ formed the independent mixture algorithm and the tree­ learning algorithm indicates that it is effectively utiliz­ ing relationships between variables, and that it includes useful relationships more complex than mere pairwise dependencies. The fact that its networks outperformed the pseudo-discrete networks and the single-Gaussian net­ works indicates that the Gaussian mixture models used for the network nodes' parameterizations helped the networks achieve much better prediction than possible with simpler parameterizations. Our primary mix-net learning algorithm took about an hour and a half of CPU time on a 400 MHz Pentium II to generate its model for each of the ten cross­ validation splits for this dataset.

The mix-net learner similarly outperformed the other al­ gorithms on the Astro dataset. The algorithm took about three hours of CPU time to generate its model for each of the cross-validation splits for this dataset.

To verify that the mix-net's performance gain over the pseudo-discrete network's was due to its use of Gaussian mixtures rather than piecewise constant densities, we con­ structed a synthetic dataset from the Bio dataset. All real values in the original dataset were discretized in a man­ ner identical to the manner in which the pseudo-discrete networks discretized them, with 16 buckets per variable. (Out of the many different numbers of buckets we tried with the pseudo-discrete networks, 16 was the number that worked best on the Bio dataset.) Each discretized value was then translated back into a real value by sampling it uniformly from the corresponding bucket's range. The re­ sulting synthetic dataset is similar in many respects to the original dataset, but its probability densities are now com­ posed of piecewise constant axis-aligned hyperboxes - precisely the kind of distributions that the pseudo-discrete networks model. The test-set performance of the pseudo­ discrete networks on this synthetic dataset is the same as their test-set performance on the original dataset (59100+/- 100). As one would expect, the test-set performance of mix-nets on this synthetic task is indeed worse (57600+/- 200). However, it is not dramatically worse.

As a second synthetic task, we generated 12,671 samples from the network learned by the Independent Mixtures al­ gorithm during one of it cross-validation runs on the Bio dataset. The test-set log-likelihood of the models learned by the Independent Mixtures algorithm on this dataset was

174 UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000

Bio Astro

|  |  |  |
| --- | --- | --- |
| Independent Mixtures | 33300 +1- 500 | 2746000 +1- 5000 |
| Single-Gaussian Mixtures | 65700 +1- 200 | 2436000 +1- 5000 |
| Pseudo-Discrete | 59100 +1- 100 | 3010000 +1- 1000 |
| Tree | 74600 +1- 300 | 3280000 +1- 8000 |
| Mix-Net | 80900 +1- 300 | 3329000 +1- 5000 |

Figure 2: Mean log-likelihoods (and the standard deviations of the means) of test sets in a 10-fold cross-validation.

32580 +1- 60, while our primary mix-net learning algorithm scored a slightly worse 31960 +1- 80. However, the net­



works learned by the mix-net learning algorithm did not actually model any spurious dependencies between vari­ ables. The networks learned by the Independent Mixtures algorithm were better only because the Independent Mix­ tures algorithm was given more time to learn each of its Gaussian mixtures.

1. POSSIBLE APPLICATIONS

Space restrictions prevent us from going into much de­ tail on possible applications; for a more extended discus­ sion, see the tech report version of this paper (Davies and Moore, 2000). The networks discussed here are trivially applicable to anomaly detection tasks in which all variables are observed. They may be well-suited to compressing datasets containing both discrete and real values (with the real values compressed lossily). It is also easy to modify our greedy network-learning algorithm to learn networks for classification tasks; the resulting networks would be similar in structure to those generated by previous algo­ rithms (Friedman et a!., 1997, 1998; Sahami, 1996), but with more flexible parameterizations.

W hile it is possible to perform exact inference in some kinds of networks modeling continuous values (e.g. (Driver and Morrell, 1995; Alag, 1996)), general-purpose exact inference in arbitrarily-structured mix-nets with continu­ ous variables may not be possible. However, inference in these networks can be performed via stochastic sampling methods (such as importance sampling or Gibbs sampling). Other approximate inference methods such as variational inference or discretization-based inference are also worth investigating.

1. CONCLUSIONS I FUTURE RESEARCH

We have described a practical method for learning Bayesian networks capable of modeling complex inter­ actions between many continuous and discrete variables, and have provided experimental results showing that the method is both feasible and effective on scientific data with dozens of variables. The networks learned by this algo­ rithm and related algorithms show considerable potential for many important applications. However, there are many

ways in which our method can be improved upon. We now briefly discuss a few of the more obvious possibilities for improvement.

* 1. VARIABLE GROUPING

The mixture tables in our network include a certain degree of redundancy, since the mixture table for each variable models the joint probability of that variable with its parents rather than just the conditional probability of that variable given its parents. One possible approach for reducing the amount of redundancy in the network is to allow each node to represent a group of variables, where each variable must

be represented in exactly one group. See the technical re­

port version of this paper (Davies and Moore, 2000) for a slightly more extensive discussion.

* 1. ALTERNATIVE STRUCTURE-LEARNERS

So far we have only developed and experimented with vari­ ations of one particular network structure-learning algo­ rithm. There is a wide variety of structure-learning algo­ rithms for discrete Bayesian networks (see, e.g., (Cooper and Herskovits, 1992; Lam and Bacchus, 1994; Heckerman et a!., 1995; Friedman et a!., 1999)), many of which could be employed when learning mix-nets. The quicker and dirtier of these algorithms might be applicable directly to learning mix-net structures. The more time-consuming al­ gorithms such as hillclimbing can be used to learn Bayesian networks on discretized versions of the datasets; the result­ ing networks may then be used as hints for which sets of dependencies might be worth trying in a mix-net. Such ap­ proaches have previously been shown to work well on real datasets (Monti and Cooper, 1998b).

* 1. ALTERNATIVE PARAMETER-LEARNERS

W hile the accelerated EM algorithm we use to learn Gaus­ sian mixtures is fast and accurate for low-dimensional mix­ tures, its effectiveness decreases dramatically as the num­ ber of variables in the mixture increases. This is the primary reason we have not yet attempted to learn mix­ ture networks with more than four variables per mixture. Further research is currently being conducted on alternate data structures and algorithms which with to accelerate EM in the hopes that they will scale more gracefully to

UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS 2000 175



higher dimensions (Moore, 2000). In the meantime, it would be trivial to allow the use of some high-dimensional single-Gaussian mixtures within mix-nets as we do for the "competing" Single-Gaussian algorithm described in Sec­ tion 4.1. Other methods for accelerating EM have also been developed in the past (e.g., (Neal and Hinton, 1998; Bradley et a!., 1998)), some of which might be used in our Bayesian network-learning algorithm instead of or in ad­ dition to the accelerated EM algorithm employed in this paper.

Our current method of handling discrete variables does not deal well with discrete variables that can take on many pos­ sible values, or with combinations involving many discrete variables. Better methods of dealing with these situations are also grounds for further research. One approach would be to use mixture models in which the hidden class vari­ able determining which Gaussian each datapoint's contin­ uous values come from also determines distributions over the datapoint's discrete values, where each discrete value is assumed to be conditionally independent of the others given the class variable. Such an approach has been used previously in AutoClass (Cheeseman and Stutz, 1996). The accelerated EM algorithm exploited in this paper would have to be generalized to handle this class of models, how­ ever. Another approach would be to use decision trees over the discrete variables rather than full lookup tables, a tech­ nique previously explored for Bayesian networks over dis­ crete domains (Friedman and Goldszmidt, 1996). In pre­ vious work (Koller et a!., 1999) employing representations closely related to the those employed in this paper, a com­ bination of these two approaches has been explored briefly in order to represent potentials in junction trees; further ex­ ploration along these lines may prove fruitful.

The Gaussian mixture learning algorithm we currently em­ ploy attempts to find a mixture maximizing the joint like­ lihood of all the variables in the mixture rather than a con­ ditional likelihood. Since the mixtures are actually used to compute conditional probabilities, some of their repre­ sentational power may be used inefficiently. The EM al­ gorithm has recently been generalized to learn joint dis­ tributions specifically optimized for being used condition­ ally (Jebara and Pentland, 1999). If this modified EM al­ gorithm can be accelerated in a manner similar to our cur­ rent accelerated EM algorithm, it may result in significantly more accurate networks.

Finally, further comparisons with alternative methods for modeling distributions over continuous variables (such as the Gaussian kernel functions used in (Hofmann and Tresp, 1995)) are warranted.

References

S. Alag. Inference Using Message Propagation and Topology Transformation in Vector Gaussian Continuous Networks. In Proc. UAI. 1996.

P. S. Bradley, U. Fayyad, and C. A. Reina. Scaling EM (Expectation-Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Re­ search, Redmond, WA, November 1998.

P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurasamy, editors, Advances in Knowledge Discovery and Data Mining. MIT Press, 1996.

D. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H.-J. Lenz, editors, Learning from Data: Artificial Intelligence and Statistics V, pages 121-130. Springer-Verlag, 1996.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, IT-14:462-467, 1968.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 9:309-347, 1992.

S. Davies and A. Moore. Bayesian Networks for Lossless Dataset Compression. In

Conference on Knowledge Discovery in Databases (KDD-99), 1999.

S. Davies and A. Moore. Mix-nets: Factored Mixtures of Gaussians in Bayesian Net­ works with Mixed Continuous and Discrete Variables. Technical Report CMU­ CS-00-119, CMU School of Computer Science, 200.

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 39:1-39, 1977.

E. Driver and D. Morrell. Implementation of Continous Bayesian Networks Using Sums of Weighted Gaussians. In Proc. UA/, 1995.

R. Duda and P. Hart. Pattern Classification and Scene Analysis. John Wiley & Sons, 1973.

N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. Machine Learning, 29:131-163, 1997.

N. Friedman, M. Godszmidt, and T. J. Lee. Bayesian Network Classification with Continuous Attributes: Getting the Best of Both Discretization and Parametric Fitting. In Proceedings of the Fifteenth International Conference on Machine Learning (/CML), 1998.

N. Friedman and M. Goldszmidt. Learning Bayesian Networks with Local Structure. In Proc. UA/, 1996.

N. Friedman, I. Nachman, and D. Peer. Learning Bayesian Network Structures from Massive Datasets: The Sparse Candidate Algorithm. In Proc. UAI, 1999.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: the combination of knowledge and statistical data. Machine Learning, 20:197-243, 1995.

D. Heckerman and C. Meek. Embedded Bayesian network classifiers. Technical Report MSR-TR-97-06, Microsoft Research, Redmond, WA, March 1997a.

D. Heckerman and C. Meek. Models and selection criteria for regression and classi­ fication. In Proc. UA/, 1997b.

R. Hofmann and V. Tresp. Discovering Structure in Continuous Variables Using Bayesian Networks. In D. S. Touretzsky, M. C. Mozer, and M. Hasselmo, editors,

Advances in Neural Information Processing Systems 8. MIT Press, 1995.

T. Jebara and A. Pentland. The Generalized CEM Algorithm. In Advances in Neural Information Processing Systems 12. MIT Press, 1999.

D. Koller, U. Lerner, and D. Angelov. A General Algorithm for Approximate Infer­ ence and Its Applications to Hybrid Bayes Nets. In Proc. UA/, 1999.

A. Kozlov and D. Koller. Nonuniform dynamic discretization in hybrid networks. In

Proc. UAI, 1997.

W. Lam and F. Bacchus. Learning Bayesian belief networks: an approach based on the MDL principle. Computational Intelligence, 10:269-293, 1994.

S. Monti and G. F. Cooper. A Multivariate Discretization Method for Learning Bayesian Networks from Mixed Data. In Proc. UA/, 1998a.

S. Monti and G. F. Cooper. Learning Hybrid Bayesian Networks from Data. In

M. I. Jordan, editor, Learning in Graphical Models. Kluwer Academic Publish­

ers, 1998b.

A. W. Moore. Very Fast EM-based Mixture Model Clustering using Multiresolution kd-trees. In Advances in Neural Processing Systems 12. MIT Press, 1999.

A. W. Moore. The Anchors Hierarchy: Using the triangle inequality to survive high dimensional data. In Proc. UA/, 2000.

A. W. Moore, J. Schneider, and K. Deng. Efficient Locally Weighted Polynomial Re­ gression Predictions. In Proceedings of the 1997 International Machine Learning Conference. Morgan Kaufmann, 1997.

R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, Learning in Graphical Models. Kluwer Academic Publishers, 1998.

M. Sahami. Learning Limited Dependence Bayesian Classifiers. In KDD-96: Pro­ ceedings of the Second International Conference on Knowledge Discovery and Data Mining, pages 335-338. AAAI Press, 1996.

P. Sand and A. W. Moore. Fast Structure Search for Gaussian Mixture Models. Submitted to Knowledge Discovery and Data Mining 200, 200.

G. Schwarz. Estimating the dimension of a model. Annals of Statistics, 6:461-464, 1978.