Discretizing Continuous Attributes While Learning Bayesian Networks

Nir Friedman
Stanford University
Dept. of Computer Science
Gates Building 1A
Stanford, CA 94305-9010
nir@cs.stanford.edu

Moises Goldszmidt
Rockwell Science Center
444 High St., Suite 400
Palo Alto, CA 94301
moises@rpal.rockwell.com

Abstract

We introduce a method for learning Bayesian networks that handles the discretization of continuous variables as an integral part of the learning process. The main ingredient in this method is a new metric based on the *Minimal Description Length* principle for choosing the threshold values for the discretization while learning the Bayesian network structure. This score balances the complexity of the learned discretization and the learned network structure against how well they model the training data. This ensures that the discretization of each variable introduces just enough intervals to capture its interaction with adjacent variables in the network. We formally derive the new metric, study its main properties, and propose an iterative algorithm for learning a discretization policy. Finally, we illustrate its behavior in applications to supervised learning.

1 INTRODUCTION

Bayesian networks provide efficient and effective representation of the joint probability distribution over a set of random variables. They have been successfully applied as expert systems, diagnostic engines, optimal decision making systems, and classifiers. Building Bayesian networks can be a laborious and expensive process in large applications. Thus, learning Bayesian networks from data has become a rapidly growing field of research that has seen a great deal of activity in recent years [1, 2, 4, 10, 14]. The objective is to induce a network (or a set of networks) that "best describes" the probability distribution over the training data. This optimization process is implemented in practice using heuristic search techniques to find the best candidate over the space of possible networks. The search process relies on a scoring metric that asses the merits of each candidate network.

In several domains of interest, such as medicine and industrial control, variables in the training data often have continuous values. We have two basic approaches to deal with continuous variables: we can restrict ourselves to specific families of parametric distributions and use the methods described in [11, 16, 12, 3], or we can discretize these variables and learn a network over the discretized domain. There is tradeoff between the two options. The first can model the conditional density of each variable in the network (under some assumptions regarding the family of distributions). However, at the current stage, we do not have efficient reasoning machinery for such models. The second only captures rough characteristics of the distribution of the continuous variables. However, it induces models that can be efficiently used for probabilistic inference and optimal decision making. Also, it is interesting to note that many of the current applications of Bayesian networks for medical diagnostic involve discretization supplied by human experts.

In this paper we present a principled approach to the second option. The problem of variable discretization is essentially that of finding for each continuous variable X, a set of *threshold* values that partition the real line into a finite number of intervals. These intervals are the values of the discretized counterpart of X. Our approach to this problem is based on the *Minimal Description Length* (MDL) principle.

The MDL score of a network B is composed of two parts. The first part measures the "complexity" of the network, while the second part measures how good the network is as a model for the data [14, 1]. The "optimal" network B given a data set D is the network that minimizes the MDL score. This minimization involves a tradeoff between the two parts of the score. In practice, the MDL score regulates the number of parameters learned and helps avoid overfitting.

We derive an augmented version of MDL to take into account the discretization of continuous variables in the data. This approach embodies a new tradeoff between the complexity of the learned discretization and the learned network, with the how well they model the training data. This tradeoff guarantees that the discretization of each variable introduces just enough intervals to capture the interaction with adjacent variables in the network. This new metric provides a principled approach for selecting the threshold values in the discretization process.

Our proposal can be regarded as a generalization of the method proposed by Fayyad and Irani [7]. Roughly speaking, their approach, which applies to supervised learning only, discretizes variables to increase the mutual information with respect to the class variable. In fact, as we show experimentally below, the two approaches are almost identical when we restrict our attention to suitable network structures where variables are restricted to interact solely through the class variable. In general, and in particular in *unsupervised learning*, we would like the discretization of each variable to maximize its mutual information with respect to *all directly related variables*. These relationships are explicitly represented by the Bayesian network structure—an arc denotes direct dependence between two variables. Thus, the local neighborhood of each variable (in the network structure) contains all the variables that directly interact with it. By having the discretization be an integral part of the learning of the Bayesian network, we are able to maximize the mutual information among related variables.

This paper is organized as follows: in Section 2 we review the MDL metric and the process of learning a Bayesian network. In Section 3 we augment this metric to account for discretizations. In Section 4 we examine computational issues and identify several properties of our scoring metric that can be exploited in this regard. In Section 5 we present experimental results that evaluate this method in classification tasks and compare it to the method proposed by Fayyad and Irani [7]. Finally, Section 6 summarizes our contributions and discusses future work.

2 LEARNING BAYESIAN NETWORKS

Consider a finite set $\mathbf{U} = \{X_1, \ldots, X_n\}$ of discrete random variables where each variable X_i may take on values from a finite domain. We use capital letters, such as X, Y, Z, for variable names and lowercase letters x, y, z to denote specific values taken by those variables. The set of values X can attain is denoted as $\mathit{Val}(X)$, the cardinality of this set is denoted as $||X|| = |\mathit{Val}(X)|$. Sets of variables are denoted by boldface capital letters $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, and assignments of values to the variables in these sets will be denoted by boldface lowercase letters $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ (we use $\mathit{Val}(\mathbf{X})$ and $||\mathbf{X}||$ in the obvious way). Let P be a joint probability distribution over the variables in $\mathbf{U}$, and let $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ be subsets of $\mathbf{U}$. $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally independent* given $\mathbf{Z}$ if for all $\mathbf{x} \in \mathit{Val}(\mathbf{X}), \mathbf{y} \in \mathit{Val}(\mathbf{Y}), \mathbf{z} \in \mathit{Val}(\mathbf{Z})$,

$$P(\mathbf{x} \mid \mathbf{z}, \mathbf{y}) = P(\mathbf{x} \mid \mathbf{z}) \text{ whenever } P(\mathbf{y}, \mathbf{z}) > 0.$$

A *Bayesian network* is an annotated directed acyclic graph that encodes a joint probability distribution of a domain composed of a set of random variables. Formally, a Bayesian network for $\mathbf{U}$ is the pair $B = \langle G, \Theta \rangle$. G is a directed acyclic graph whose nodes correspond to the random variables $X_1, \ldots, X_n$, and whose edges represent direct dependencies between the variables. The graph structure G encodes the following set of independence assumptions: each node X_i is independent of its non-descendants given its parents in G [17].[1] The second component of the pair, namely Θ, represents the set of parameters that quantifies the network. It contains a parameter $\theta_{x_i|\Pi_{x_i}} = P(x_i|\Pi_{x_i})$ for each possible value x_i of X_i, and Π_{x_i} of Π_{X_i} (the set of parents of X_i in G). B defines a unique joint probability distribution over $\mathbf{U}$ given by:

$$P_B(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i|\Pi_{X_i}) = \prod_{i=1}^{n} \theta_{X_i|\Pi_{X_i}} \quad (1)$$

The problem of learning a Bayesian network can be stated as follows. Given a *training set* $D = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ of instances of $\mathbf{U}$ (i.e., each $\mathbf{u}_i$ is a value assignment to all variables in $\mathbf{U}$), find a network B that *best matches* D. To formalize the notion of goodness of fit of a network with respect to the data, we normally introduce a scoring function, and to solve the optimization problem we usually rely on heuristic search techniques over the space of possible networks [9]. Several different scoring functions have been proposed in the literature [4, 10, 14]. In this paper we focus our attention on the MDL score [14]. This score is simple, very intuitive, and has proven to be quite effective in practice.

The idea behind the MDL principle is as follows. Suppose that we are given a set D of instances which we would like to store and keep in our records. Naturally, we would like to conserve space and save a compressed version of D. To this end we need to find a suitable model for D such that an encoder can take this model and produce a compact image of D. Moreover, as we want to be able to recover D, we must also store a version of the model used by the encoder to compress D. The description length of the data based on a model, and using a particular encoder, is then the length of the compressed data plus the representation size of the model itself. In our case the length is measured by the number of bits needed for storage. The MDL principle dictates that the optimal model is the one (from a particular class of interest) that minimizes the total description length.

The MDL principle is applied to learning Bayesian networks by taking a network to be the model for the data used by an encoder to produced a compressed version of D. The idea is as follows: a network B assigns a probability to each instance of $\mathbf{U}$. Using these probabilities we can construct an efficient code. In particular, we use the Huffman code [5], which assigns shorter codes to frequent instances. The benefit of using the MDL as a scoring metric is that the best network for D optimally balances the complexity

[1] Formally there is a notion of minimality associated with this definition, but we will ignore it in this paper. See [17] for details.

of the network with the degree of accuracy with which the network represents the frequencies in D.

We now describe in detail the representation length required for the storage of both the network and the coded data. The MDL score of a candidate network is defined as the total description length. To store a network $B = \langle G, \Theta \rangle$, we need to describe $\mathbf{U}$, G, and Θ:

To describe $\mathbf{U}$, we store the the number of variables, n, and the cardinality of each variable X_i. This information can be stored in $\log n + \sum_i \log ||X_i||$ bits.[2]

To describe the DAG G it is sufficient to store for each variable X_i a description of Π_{X_i} (namely, its parents in G). This description consists of the number of parents followed by a list of the parents. Since we can encode each of these using $\log n$ bits, the graph structure can be encoded in $\sum_i (1 + |\Pi_{X_i}|) \log n$ bits.

To describe the parameters in Θ, we must consider the parameters in each conditional probability table. For the table associated with X_i, we need to store $||\Pi_{X_i}||(||X_i|| - 1)$ parameters. The representation length of these parameters depends on the number of bits we use for each numeric parameter. The usual choice in the literature is $1/2 \log N$ [1, 9]. Thus, the encoding length of Θ is $\frac{1}{2} \log N \sum_i ||\Pi_{X_i}||(||X_i|| - 1)$

We remark that the description length of B is only a function of $\mathbf{U}$ and of G, and it does not depend on the actual values of parameters in Θ. Combining all three components we have:

$$
\begin{aligned}
\mathit{DL}_{\mathit{net}}(U, G) \quad = \quad & \sum_i \big(\log ||X_i|| + (1 + |\Pi_{X_i}|) \log n \big) + \\
& \frac{\log N}{2} \sum_i ||\Pi_{X_i}||(||X_i|| - 1)
\end{aligned}
$$

We turn our attention to the description length of the data. Using the probability measure defined by B, we construct a Huffman code for the instances in D. In this code, the exact length of each codeword depends on the probability assigned to that particular instance. There is no closed form description of this length. However, it is known [5] that when we choose longer coding blocks we can approximate the optimal encoding length in which the encoding of each $\mathbf{u}$ is $-\log P_B(\mathbf{u})$ bits. Thus, the description length of the data is simply: $-\sum_{i=1}^{N} \log P_B(\mathbf{u}_i)$

We can rewrite this expression in a more convenient form. Let $\hat{P}_D$ be the probability measure that corresponds to the frequency of instances in the data. Using (1) and some algebraic manipulations we can rewrite the representation length of the data as:

$$
-\sum_{i=1}^{N} \log P_B(\mathbf{u}_i) = -N \sum_i \sum_{x_i, \Pi_{x_i}} \hat{P}_D(x_i, \Pi_{x_i}) \log \theta_{x_i | \Pi_{x_i}}
$$

Using standard arguments, we can show that this term is minimized if we set $\theta_{x_i|\Pi_{x_i}} = \hat{P}_D(x_i | \Pi_{x_i})$. From now on, we assume that we will always choose the values of Θ in this manner. Thus, we can write the description length of the data, given a network structure G, as:

$$
\mathit{DL}_{\mathit{data}}(D, G) = N \sum_i H(X_i | \Pi_{X_i})
$$

Where $H(\mathbf{X}|\mathbf{Y}) = -\sum_{\mathbf{x},\mathbf{y}} \hat{P}_D(\mathbf{x}, \mathbf{y}) \log \hat{P}_D(\mathbf{x}|\mathbf{y})$ is the *conditional entropy* of $\mathbf{X}$ given $\mathbf{Y}$ (if $\mathbf{Y} = \emptyset$ then $H(\mathbf{X}|\mathbf{Y}) = H(\mathbf{X})$, the unconditional entropy of $\mathbf{X}$) [5]. Using this information-theoretic interpretation we can rewrite $\mathit{DL}_{\mathit{data}}(D, G)$ slightly. The *mutual information* between two sets of variables $\mathbf{X}$ and $\mathbf{Y}$ is defined as $I(\mathbf{X}; \mathbf{Y}) = \sum_{\mathbf{x},\mathbf{y}} \hat{P}_D(\mathbf{x}, \mathbf{y}) \log \frac{\hat{P}_D(\mathbf{x},\mathbf{y})}{\hat{P}_D(\mathbf{x})\hat{P}_D(\mathbf{y})}$. (When $\mathbf{Y} = \emptyset$ we define $I(\mathbf{X}; \mathbf{Y}) = 0$.) It is easy to verify that $H(\mathbf{X}|\mathbf{Y}) = H(\mathbf{X}) - I(\mathbf{X}; \mathbf{Y})$. Thus, we can rewrite:

$$
\mathit{DL}_{\mathit{data}}(D, G) = N \sum_i H(X_i) - N \sum_i I(X_i; \Pi_{X_i})
$$

Since $H(X_i)$ does not depend on the structure G, the first term is constant in the description length of all possible structures. Thus, we ignore it when we compare candidate network structures. This information-theoretic interpretation shows that to minimize the description length of D we need to find a structure that maximizes the mutual information between each variable and its parents.

Finally, the MDL score (of a candidate network structure G) is defined as the total description length

$$
\mathit{DL}(U, G, D) = \mathit{DL}_{\mathit{net}}(U, G) + \mathit{DL}_{\mathit{data}}(D, G) \qquad (2)
$$

According to the MDL principle, we should strive to find the network structure that minimizes the description length. In practice, this is usually done by searching over the space of possible networks. Since this space is large, this search is a non-trivial problem. Details about the search process can be found in [1, 9].

3 MDL SCORE FOR DISCRETIZATION

Until now we have assumed that all the variables X_i in the universe $\mathbf{U}$ are discrete (or nominal). We now consider the case where some of the variables take numerical values. We assume that $\mathbf{U}_{\mathit{cont}} \subseteq \mathbf{U}$ is the set of continuous variables in $\mathbf{U}$.

A *discretization sequence* $\lambda = \langle t_1, \ldots, t_k \rangle$ is an increasing sequence of real values (i.e., $t_1 < t_2 < \ldots < t_k$). Such

[2]This number is slightly imprecise since we do not know in advance the length of the encoding [5].

a sequence defines a function $f_\lambda : I\!R \mapsto \{0, \ldots, k\}$ as follows:

$$f_\lambda(x) = \begin{cases} 0 & \text{if } x < t_1 \\ i & \text{if } t_i \leq x < t_{i+1} \text{ for } 1 \leq i < k \\ k & \text{if } t_k \leq x \end{cases}$$

A *discretization policy* is a collection $\Lambda = \{\lambda_i : X_i \in \mathbf{U}_{cont}\}$. Such a policy defines a new collection of variables $\mathbf{U}^*_\Lambda = \{X^*_i : X_i \in \mathbf{U}\}$, where $X^*_i = f_{\lambda_i}(X_i)$ if $X_i \in \mathbf{U}_{cont}$ and $X^*_i = X_i$ otherwise. Thus, X^*_i is the discrete random variable associated with X by the discretization policy Λ. From now on we will omit the subscript Λ when it is understood from the context.

Our target is to (jointly) learn a discretization policy and a Bayesian network over the discretized data. To do so, we augment the MDL score to include the description length needed for storing enough information to be able to recover the original data from the discretized data.

Lets assume (for now) that both G and Λ are fixed. Let D be a collection of instances over $\mathbf{U}$. We define D^* to be the discretization of D using Λ. Namely, $D^* = \{\mathbf{u}^*_1, \ldots, \mathbf{u}^*_N\}$, where if $\mathbf{u}_j = \langle x^{(j)}_1, \ldots, x^{(j)}_n \rangle$, then $\mathbf{u}^*_j = \langle f_{\lambda_1}(x^{(j)}_1), \ldots, f_{\lambda_n}(x^{(j)}_n) \rangle$. The output of our encoder consists of the following parts:

1. The description of D^* (using a network based on G).
2. The description of Λ.
3. The description of D based on D^*, e.g., an encoding of the information necessary to recover each $\mathbf{u}_j$ from $\mathbf{u}^*_j$.

Since D^* is a discrete dataset, the description length of the first part is the one derived in Section 2, and $\mathit{DL}(U^*, G, D^*)$ bits long (Eq. 2).

We introduce some useful notation. Let $\mathit{Val}_D(X_i)$ to be the set of values of X_i in that appear in the dataset D, and let $N_i = |\mathit{Val}_D(X_i)|$ to be the number of these values (clearly $N_i \leq N$). Since we need to recover D we have to record the set of values $\mathit{Val}_D(X_i)$ for each continuous random variable $X_i \in \mathbf{U}_{cont}$. The representation of this set depends on the exact precision in which we choose to encode real numbers. However, since this encoding depends only on D and not on Λ nor G, it has the same length for all possible instances of G and Λ. Thus, we will ignore this factor in the total score.

We now turn to the description of the discretization policy Λ. The description of $\mathbf{U}^*$ (which is stored as part of the description of D^*), already records the number of values of X^*_i. Thus we already recorded the number of threshold points, and only need to encode the threshold values in λ_i. Since we only need to distinguish among values in $\mathit{Val}_D(X_i)$, we can limit our attention to threshold values that are mid-points between successive values in this set. Thus, the thresholds t_j's are chosen from among the $N_i - 1$ mid-point values in $\mathit{Val}_D(X_i)$. All we need to store is the index of the discretization policy in some enumeration of the $\binom{N_i-1}{k_i-1}$ different discretization policies of cardinality $k_i = ||X^*_i||$. This index can be described using $\log \binom{N_i-1}{k_i-1}$ bits. Using Sterling's approximation (see [5, p. 151]), we get that $\log \binom{N_i-1}{k_i-1} \leq (N_i - 1)H(\frac{k_i-1}{N_i-1})$, where $H(p) = -p \log p - (1-p)\log(1-p)$.[3] Thus the description of Λ is of length:

$$\mathit{DL}_\Lambda(\Lambda) = \sum_{X_i \in \mathbf{U}_{cont}} (N_i - 1) H(\frac{k_i - 1}{N_i - 1})$$

Finally, we need to describe how to reconstruct each $\mathbf{u}_j$ from $\mathbf{u}^*_j$. For each continuous X_i, the value of X^*_i in the discretized instance $\mathbf{u}^*_j$ specifies an interval. We need only to encode what value in this interval actually appears in $\mathbf{u}_j$. Note that there is a limited number of such values in $\mathit{Val}_D(X_i)$. Thus, we can use a Huffman code to encode which one of those appears in the specific instance $\mathbf{u}_j$. The optimal Huffman code is the one constructed according to the frequencies of values in $\mathit{Val}_D(X_i)$. Thus, we need to store these frequencies as well. Since these frequencies depend only on the data D, and not on the particular discretization policy or network structure, we can ignore them in the description length.

Given that we have recorded these frequencies, we can reconstruct the code for each value of X^*_i. The encoding for a particular value of X_i is approximately $-\log \hat{P}_D(X_i | X^*_i)$ bits long. Summing the length of the encoding for all instances in D and for all variables in $\mathbf{U}_{cont}$, we get:

$$\begin{aligned} \mathit{DL}_{D^* \to D}(D, \Lambda) &= -\sum_i \sum_{j=1}^{N} \log \hat{P}_D(x^{(j)}_i | \lambda_i(x^{(j)}_i)) \\ &= N \sum_i H(X_i | X^*_i). \end{aligned} \quad (3)$$

Combining all these parts together, we conclude that the representation length of D, using Λ and G, is:

$$\mathit{DL}(\mathbf{U}^*, G, D^*) + \mathit{DL}_\Lambda(\Lambda) + \mathit{DL}_{D^* \to D}(D, \Lambda).$$

We now rewrite this term to a more convenient expression. Using basic properties of the entropy and mutual information measures (see [5]), we easily prove the following:

Proposition 3.1: $H(X^*_i | \Pi^*_{X_i}) + H(X_i | X^*_i) = H(X_i) - I(X^*_i ; \Pi^*_{X_i})$

Note that $H(X_i)$ is a function of the data D and does not depend on Λ and G. Thus, we can ignore it. With this simplification we take the score to be:

$$\begin{aligned} \mathit{DL}^*(G, \Lambda, D) &= \mathit{DL}_{net}(\mathbf{U}^*, G) + \mathit{DL}_\Lambda(\Lambda) \\ &\quad - N \sum_i I(X^*_i ; \Pi^*_{X_i}) \end{aligned} \quad (4)$$

[3] This bound is tight in the following sense: $(N_i - 1)H(\frac{k_i-1}{N_i-1}) - \log(N_i) \leq \log \binom{N_i-1}{k_i-1}$

The only difference with the previous version of the score represented by Eq. 2 is the middle term which encodes the discretization policy. Thus, as in the previous case, the score weights the addition of a new arc (an increase in the complexity of the network) against the increase in mutual information between the nodes in the extremes of this edge. The additional term, $DL_\Lambda(\Lambda)$, takes into account another tradeoff involving the number of intervals in the discretization against the mutual information score.

4 LEARNING DISCRETIZATIONS

We now dispense with the assumption in the previous section that both Λ and G are set, and examine how to learn a discretization policy and network structure in practice. We will start by assuming that we have a fixed network structure G, and examine how to find the discretization policy Λ that minimizes the score $DL^*(G, \Lambda, D)$. We consider first the simplest case where there is exactly one continuous variable in **U**, and examine a number of properties that can be exploited in the computation of $DL^*(G, \Lambda, D)$. For the general case, where more than one variable is discretized, we will describe an iterative approach that takes advantage of these results. Finally, in Section 4.3 we discuss how to learn the network structure as well as a the discretization policy.

4.1 DISCRETIZING ONE VARIABLE

Let X_i be the continuous variable to be discretized. We examine how $DL^*(G, \Lambda, D)$ changes as we change Λ. As we will show most of the terms in this score do not change when we change the discretization of X_i. This fact can be exploited to reduce the computations during the discretization process.

We define the *local score* of X_i to consist of the terms in $DL^*(G, \Lambda, D)$ that involve X_i's discretization:

$$\begin{aligned} &DL_{local}(X_i; G, \Lambda, D) = \\ &\quad \tfrac{1}{2}\log N \cdot \|\Pi^*_{X_i}\|(\|X^*_i\| - 1) \\ &+ \quad \tfrac{1}{2}\log N \cdot \textstyle\sum_{j, X_i \in \Pi_{X_j}} \|\Pi^*_{X_j}\|(\|X^*_j\| - 1)] \\ &+ \quad \log k_i + (N_i - 1) H\left(\tfrac{k_i - 1}{N_i - 1}\right) \\ &- \quad N \cdot [I(X^*_i; \Pi^*_{X_i}) + \textstyle\sum_{j, X_i \in \Pi_{X_j}} I(X^*_j; \Pi^*_{X_j})] \end{aligned}$$

Immediately we get:

Proposition 4.1: *Let Λ and Λ' be two discretization policies that differ only on the discretization of X_i. Then $DL^*(G, \Lambda, D) - DL^*(G, \Lambda', D) = DL_{local}(X_i; G, \Lambda, D) - DL_{local}(X_i; G, \Lambda', D)$.*

Thus, when we are only changing the discretization of X_i, it suffices to minimize $DL_{local}(X_i; G, \Lambda, D)$. The important aspect of this result, is that $DL_{local}(X_i; G, \Lambda, D)$ involves only variables that are in the *Markov blanket* of X_i [17], i.e., X_i's parents, X_i's children, and parents of X_i's children in G. The term we attempt to minimize weights the cost of describing k partitions for X_i against the mutual information gained about the parents and children of X_i from having this number of partitions.

The problem now is how to search for a discretization that minimizes this DL_{local}. As we noted above, the only points that we should consider for thresholds are the $N_i - 1$ midpoints between successive values of X_i. The space of possible discretizations consists then of all subsets of these $N_i - 1$ points. In practice, we search this space by starting with the trivial discretization (i.e., the discretization with the empty threshold list), and search over the possible refinements. This approach, which is usually called *top-down*, is common in the supervised learning literature [18, 7]. The search strategy can be any of the well known ones, greedy search (or hill climb search), beam search, etc.

Carrying out this search can be very expensive. Even for a simple greedy search strategy we need to perform $N_i k - \frac{1}{2}k(k+1)$ threshold evaluations to find a discretization with k thresholds. However, as we show below, because of some properties of DL_{local} it is not always necessary to recompute the change of DL_{local} for all possible splits at each step.

Let $\lambda = \langle t_1, \ldots, t_k \rangle$. Define $\lambda \cdot t$ to be the *refinement* of λ, e.g., the discretization sequence that contains $t_1, \ldots, t_k$ and t. Suppose our current candidate for discretizing X_i is λ. The possible successors are of the form $\lambda \cdot t$, where t is one of the midpoints of $Val_D(X_i)$. We evaluate each successor t by examining the change in the term DL_{local} described above. We notice, however, that the only term in DL_{local} that depends on the particular threshold we choose to add is the one that measures the mutual information between X_i and its Markov blanket. Thus, to find the best successor of a discretization sequence, we need only to compute the *information gain* of each possible successor:

$$\begin{aligned} &Gain(t; \lambda, X, G, D) = \\ &\qquad I(X^*_{\lambda \cdot t}; \Pi^*_X) + \textstyle\sum_{Y, X \in \Pi_Y} I(Y^*; \Pi^*_{Y\,\lambda \cdot t}) \\ &- \quad I(X^*; \Pi_X) - \textstyle\sum_{Y, X \in \Pi_Y} I(Y^*; \Pi^*_Y) \end{aligned}$$

This term has the following property that we can use to speed up the search:

Proposition 4.2: *If $f_\lambda(t) \neq f_\lambda(t')$, then $Gain(t'; \lambda \cdot t, X, G, D) = Gain(t'; \lambda, X, G, D)$.*

Using this proposition we conclude that if we choose to add the threshold t to λ, we only need to recompute the gain for those thresholds that split the same interval that t splits.

Our implementation uses a greedy search routine. This routine maintains a list of possible splits and their associated gain. At each iteration the best split is added to the current candidate. If the change in description length is negative the routine terminates. Otherwise, it recomputes the gain only for the relevant thresholds (according to Proposition 4.2) and reiterates.

4.2 DISCRETIZING SEVERAL VARIABLES

We now turn to the more general problem where there are several continuous variables in $\mathbf{U}$. Computationally, this seems to be a hard problem. We need to search over all possible discretizations for all variables in $\mathbf{U}_{cont}$. However, we can take advantage of the structure of G and the independences it encodes to find *non-interacting* discretizations. Roughly speaking, X and Y are non-interacting if we can discretize each one of them separately. Let Λ be some discretization policy, let $\mathbf{X} \subseteq \mathbf{U}_{cont}$, and let $\Lambda_{\mathbf{X}}$ to be a discretization policy over X. We define $\Lambda[\Lambda_{\mathbf{X}}]$ to be the discretization policy that results by replacing the discretization of each $X \in \mathbf{X}$ in Λ by the discretization of X in $\Lambda_{\mathbf{X}}$.

Definition 4.3: Let $\mathbf{X}, \mathbf{Y}$ be two disjoint subsets of $\mathbf{U}_{cont}$. $\mathbf{X}$ and $\mathbf{Y}$ are *non-interacting* (with respect to a DAG G) if for all set of instances D, discretization policies Λ and discretization policies $\Lambda_{\mathbf{X}}$ and $\Lambda_{\mathbf{Y}}$ for $\mathbf{X}$ and $\mathbf{Y}$, we have that $DL^*(G, \Lambda[\Lambda_{\mathbf{X}}, \Lambda_{\mathbf{Y}}], D) = DL^*(G, \Lambda[\Lambda_{\mathbf{X}}], D) + DL^*(G, \Lambda[\Lambda_{\mathbf{Y}}], D) - DL^*(G, \Lambda, D)$. ∎

Thus, X and Y are non-interacting if the difference in the total score when we change the discretization of both X and Y is the sum of differences in the score when we change each one individually. In particular, this implies that we can optimize the discretization for X and for Y independently of each other. On the other hand, when X and Y are interacting, it might be the case that after discretizing one set, we might have to reconsider the discretization of the other set.

We can read many non-interaction relationships from the structure of G. The *moral-graph* of $G = (\mathbf{U}, E)$ is an undirected graph $M(G)$, such that X is adjacent to Y in $M(G)$ if one of the following cases hold: $(X, Y) \in E$, $(Y, X) \in E$, or there is some Z such that $(X, Z), (Y, Z) \in E$. It is easy to verify that the Markov blanket of X is exactly the set of adjacent nodes of X in $M(G)$. Since the discretization of X depends only on its Markov blanket we can show:

Theorem 4.4: *Let* $\mathbf{X}, \mathbf{Y}$ *be two disjoint subsets of* $\mathbf{U}_{cont}$. *If* $\mathbf{X}$ *is not adjacent to* $\mathbf{Y}$ *in* $M(G)$ *then* $\mathbf{X}$ *and* $\mathbf{Y}$ *are non-interacting with respect to* G.

Example 4.5: Let $\mathbf{U} = \{A_1, \ldots, A_n, C\}$, where the variables $A_1, \ldots, A_n$ are the *attributes* and C is the *class* variable. A *naive Bayes* model is Bayesian network with the following structure: $\Pi_C = \emptyset$, i.e., the class variable is the root; and $\Pi_{A_i} = \{C\}$ for all $1 \leq i \leq n$, i.e., the only parent for each attribute is the class variable. Then, each attribute's Markov blanket consists only of the class variable, and the attributes are non-interacting. Consequently we can discretize each one in isolation, using the techniques introduced in the previous section, taking into account only the class variable. ∎

From now on we assume that if $\mathbf{X}$ and $\mathbf{Y}$ are adjacent in $M(G)$, then they actually interact. We say that a set $\mathbf{X} \subseteq \mathbf{U}_{cont}$ is an *interacting component* (of G), if for all $\mathbf{Y} \subset \mathbf{X}$, $\mathbf{Y}$ and $\mathbf{X} - \mathbf{Y}$ are interacting. Intuitively, an interacting component cannot be subdivided into non-interacting subsets. Finding the interacting components in G can be done very efficiently (formally this is equivalent to finding connected components in a graph).

Our definition of non-interaction guarantees that we can discretize the variables in each interacting component independently of the discretization of other interacting components. If all interacting components are singletons, then the problem reduces to discretizing single variables as discussed in the previous section.

The question is how deal with non-trivial interacting components. Suppose X and Y interact. Then, changing the discretization of X changes the local environment of Y. This might lead us to reconsider the discretization of Y. After doing so, we might need to reconsider the discretization of X, and so forth. There are two main approaches to deal with this problem. The first is to search the space of possible discretizations of all the variables in the interacting component at once. This approach ensures that we consider all the interactions in the component while we discretize the variables in it. The problem is that for large components this procedure is impractical.

The second approach is to discretize one variable at a time, treating all other variables as being discrete and fixed. Roughly speaking this procedure goes at follows: we find some initial discretization for each variable. Then at each step we select a variable X_i and search for a discretization of X_i while treating all other variables as discrete (i.e., their discretization policy is held fixed). This can be done using the method described in the previous section. We repeat this step until we cannot improve the discretization of any of the variables in the component. It is easy to see that this approach must converge. At each step we are improving the overall score of the current discretization policy. And, since the score function is bounded from below, we must stop at some stage. We remark that this procedure is essentially a hill-climbing procedure and only guarantees to find a local minima of the score and not necessarily the optimal discretization. A remaining question is the order in which variables are discretized. In our experiments we used the algorithm outlined in Figure 1, which seemed to perform reasonably well. This algorithm has the desirable property that between any two discretization passes on X_i, the discretizations of all the variables in X_i's Markov blanket are reconsidered. This guarantees that we do not discretize X_i before we propagate the changes made in the previous discretization of X_i to all of its neighbors.

This procedure has to be initialized with some discretization. In our current implementation, we use the *least square quantization* [5] method. This method attempts to find the best k-partitioning of a random variable, essentially by

Input: An initial discretization Λ of **U**.
Output: A (locally) optimal discretization of **U**.
Push all continuous variables onto a queue Q.
While Q is not empty
Remove the first element X from Q.
Compute λ'_X, a new discretization of X.
If $DL^*(G, \Lambda[\lambda'_X], D) < DL^*(G, \Lambda, D)$ then
Replace λ_X by λ'_X in Λ.
For all Y interacting with X, if $Y \notin Q$, push Y onto Q.
return Λ.

Figure 1: *Algorithm for discretizing several variables in an iterative fashion.*

matching k Gaussians to the distribution of the values. In our experiments we initialized the discretization with the minimum between $k = log N_i$ and an upper bound of 5. This choice, however, however is quite arbitrary. We suspect that the finding a good initial discretization can greatly improve the quality of the discretization found. We are currently exploring this issue in more detail.

4.3 LEARNING THE NETWORK STRUCTURE

In many situations we need to learn the structure of the network G. This is once more an optimization problem; we want to find a network structure G and a discretization policy Λ to maximize $DL^*(G, \Lambda, D)$.

There are two possible ways of going about this. The first is very similar to current approaches to learning Bayesian networks. In this approach we search over the space of possible candidates for G. To evaluate each candidate, we search for a discretization Λ that maximizes the score given that candidate. This search procedure is computationally costly, even though we can use the local properties of the discretization to avoid rediscretizing all the variables in each candidate network.

The second approach alternates between learning the discretization policy and learning the network structure in an iterative fashion. The idea is as follows: we start with some discretization, and learn a network structure given this (fixed) discretization. Then, we rediscretize based on the learned network. This cycle is repeated until no improvement is made. Again, since the procedure improves the score on each iteration, it is easy to see that this procedure must terminate. (In our experiments this process almost always terminated in less than 3 iterations.) This procedure, however, only guarantees to find a a local minima, which is not necessarily an optimal solution. As we will show below, the choice of the initial discretization can affect the outcome of the procedure. We are currently exploring these issues in more detail.

Table 1: Experimental results comparing naive Bayes models to the method of Fayyad and Irani.

Dataset	Prediction Accuracy	
	naive	**FI-naive**
australian	**86.38+-1.38**	86.23+-1.10
breast	**97.36+-0.55**	**97.36+-0.55**
cleve	**82.76+-1.66**	**82.76+-1.27**
crx	**86.84+-1.10**	86.22+-1.14
diabetes	**74.48+-0.76**	**74.48+-0.89**
glass	62.64+-1.86	**67.78+-2.17**
glass2	76.10+-1.68	**79.77+-1.50**
iris	**94.00+-1.25**	**94.00+-1.25**
lymphography	**81.72+-2.62**	**81.72+-2.62**
pima	**75.65+-1.23**	75.51+-1.63
shuttle-small	98.50+-0.28	**98.76+-0.25**
vehicle	58.75+-1.43	**58.99+-1.57**
waveform-21	78.17+-0.60	**78.68+-0.60**

5 PRELIMINARY EXPERIMENTAL RESULTS

This section describes preliminary experiments designed to test the soundness of the method proposed. The experiments were run on 13 datasets from the Irvine repository [15]. We estimated the accuracy of the learned classifiers using 5-fold cross-validation, except for the "shuttle-small" and "waveform-21" datasets where we used the hold-out method. We report the mean of the prediction accuracies over all cross-validation folds. We also report the standard deviation of the accuracies found in each fold. These computations were done using the MLC++ library. (See [8, 13] for more details.)

Our first experiment is concerned with an application to supervised learning and a comparison to the discretization method of Fayyad and Irani [7] (FI from now on). This method is considered state of the art in supervised learning [6]. In their approach, FI attempt to maximize the mutual-information between each variable and the class variable. Although their method was not developed in the context of Bayesian networks, it is applicable for one particular network structure, namely that of a naive Bayes classifier. As Example 4.5 shows, in the naive Bayes classifier, the Markov blanket of each attribute consists of the class node only. In this network, our approach will also discretize each variable to maximize the mutual information with the class variable. Therefore, in this case, under this particular fixed structure for the network, it is reasonable to compare our method to theirs.

Table 1 shows the results of this comparison: **naive** denotes a naive Bayes classifier that was learned from the original data discretized with our approach; in **FI-naive** the data set was prediscretized by the method of FI using only the training data, in the manner described in [6], then, a naive Bayes classifier was learned over the discretized data. As these

Table 2: Experimental results comparing unsupervised learning of network structure.

Dataset	Prediction Accuracy		
	unsup(LS)	**unsup(naive)**	**FI-unsup**
australian	86.52+-1.42	**86.81+-0.62**	86.09+-1.58
breast	**97.51+-0.50**	96.78+-0.68	96.92+-0.49
cleve	**82.44+-1.65**	82.08+-1.51	80.73+-1.31
crx	**86.84+-1.42**	85.91+-0.92	84.38+-0.90
diabetes	74.74+-0.60	**76.30+-0.41**	75.91+-0.29
glass	52.41+-5.11	**60.80+-5.51**	55.57+-5.39
glass2	70.55+-3.28	**75.49+-3.13**	**75.49+-2.47**
iris	**94.67+-1.33**	94.00+-1.25	94.00+-1.25
lymphography	77.70+-2.74	**77.72+-1.65**	75.01+-2.69
pima	**76.30+-1.17**	75.39+-1.59	74.74+-1.24
shuttle-small	**99.17+-0.21**	96.07+-0.44	**99.17+-0.21**
vehicle	58.75+-1.56	**63.60+-2.24**	60.99+-1.94
waveform-21	**75.47+-0.63**	68.45+-0.68	69.85+-0.67

results show, FI's method performed slightly better—which is not surprising given the that it was specially designed for classification problems. Yet, our method is comparable and the discretizations found often coincided.

In the first experiment we used a fixed network structure. In the second, we learn both the discretization and the structure of the network to examine the effects of the interaction. We learned structure by searching in a greedy fashion over the space Bayesian networks. This procedure is unsupervised, *it does not distinguish the class variable from other variables in the domain.*[4] Table 2 contains the results of this experiment: **unsup(LS)** denotes the unsupervised learning method described in Section 4.3, where the initial discretization was performed by least square quantization as described in Section 4.2; **unsup(naive)** denotes an unsupervised learning method where the initial discretization is the one found by the **naive** procedure above; in **FI-unsup** the unsupervised learning was performed on training data prediscretized using FI's method. These results are inconclusive. We might have expected that **unsup(naive)** would be better since it is biased toward better classification. Yet, somewhat surprisingly there is no clear dominance between **unsup(LS)** and **unsup(naive)**. It is evident, though, that FI's method, which is informed about the classification task, is not the optimal discretization for the networks learned. This is due to the fact that it does not take into account the interactions between the other variables in the discretization process.

Finally in order to compare the discretization policies computed in each of these experiments, we used the discretizations learned by **naive**, **unsup(LS)**, **unsup(naive)** and FI's method as a prediscretized input for the C4.5 classifier [18]. Table 3 lists the prediction accuracies as well as C4.5's performance when given the data without prediscretization. As these results show that our procedures, even the unsupervised ones, are comparative with FI's method and with C4.5's internal discretization.

We emphasize that these results compare a supervised approach, i.e., one that is informed about the goal of the discretization process, with an unsupervised approach that does not give the class variable any special status. Yet, the results show that our unsupervised method is competitive with a state-of-the-art supervised method. The task we examined here is a very narrow one. It measures in a very limited fashion the quality of the learned model. We are currently devising experimental setup to compare our approach with parametric method for dealing with continuous variables.

6 DISCUSSION

This paper presents an innovative approach to the discretization of continuous variable in supervised and unsupervised learning based on the MDL principle. Discretization is handled as an integral part of the learning of a Bayesian network. This has a number of advantages. The first is that the discretization of each variable introduces just enough intervals to capture the interactions with adjacent variables in the network. The second is that non-interacting variables can be discretized in isolation. We have validated our approach against a state-of-the-art discretization procedure in the context of supervised learning, and are currently performing experiments to validate the approach in applications involving unsupervised learning. One problem we are facing is that since this is the the first approach to discretizing while learning Bayesian networks, we lack a methodological reference and established validation standards. We are currently in the process of defining these.

In addition, this research opened a number of issues that we are currently investigating. The first one is understanding how our iterative procedures depend on the initial conditions and what are good choices for these initial conditions. The second one is how to scale up the process for large databases. Currently our discretization procedure tests each mid-point among the continuous values. When the data contains a large number of values for a continuous variable this procedure becomes extremely expensive. We are considering several approaches including subsampling techniques and simple quantization methods for finding a reasonably sized candidate pool for threshold values.

Finally, an open question is the development of a similar scoring metric for discretization based on Bayesian concepts as opposed to the MDL principle. A natural path would be to augment the Bayesian scoring metric introduced in [11, 10]. The main obstacle is to specify in a compact way a prior for the parameters Θ, for each possible discretization of the data. This is a non trivial problem,

[4] We note that unsupervised Bayesian network classifiers are often better than the naive Bayes classifier. However, in datasets with multiple attributes their performance can be poor. We explain this phenomena is detail in [8].

Table 3: Using learned discretizations for C4.5.

Dataset	Prediction Accuracy				
	naive	**unsup(LS)**	**unsup(naive)**	**FI**	**C4.5**
australian	**85.94+-1.66**	68.70+-17.21	85.65+-1.51	85.65+-1.82	85.36+-0.74
breast	95.17+-0.59	95.02+-0.59	95.02+-0.43	94.73+-0.59	**95.32+-0.44**
cleve	72.97+-0.54	**78.71+-0.46**	77.36+-1.19	73.31+-0.63	74.29+-2.75
crx	85.91+-0.94	85.00+-1.08	**86.22+-1.36**	**86.22+-0.58**	84.53+-0.93
diabetes	75.00+-1.36	73.18+-1.27	75.13+-1.17	**76.04+-0.85**	70.84+-1.67
glass	66.36+-1.56	65.45+-3.94	69.19+-2.62	**69.62+-1.95**	67.75+-2.50
glass2	75.44+-1.10	74.26+-4.24	74.28+-3.60	**76.67+-1.63**	73.60+-4.06
iris	94.00+-1.25	**95.33+-0.82**	94.00+-1.25	94.00+-1.25	94.67+-1.33
lymphography	77.03+-1.21	**77.70+-0.82**	74.97+-1.84	77.03+-1.21	77.01+-0.77
pima	**75.65+-1.29**	74.99+-1.14	75.39+-1.12	75.13+-1.52	72.65+-1.78
shuttle-small	99.22+-0.20	99.38+-0.18	99.33+-0.19	99.17+-0.21	**99.53+-0.15**
vehicle	69.16+-2.30	66.31+-1.57	**70.80+-1.02**	69.74+-1.52	69.86+-1.84
waveform-21	72.47+-0.65	69.98+-0.67	72.64+-0.65	**74.70+-0.63**	70.36+-0.67

for which the methods of [10] do not immediately apply.

Acknowledgments

The authors are grateful to Denise Draper, Usama Fayyad, Ronny Kohavi and Meheran Sahami for comments on a previous draft of this paper and useful discussions relating to this work. Parts of this work were done while the first author was at Rockwell Science Center. The first author was also supported in part by an IBM Graduate fellowship and NSF Grant IRI-95-03109.

References

[1] R. R. Bouckaert. Properties of Bayesian network learning algorithms. In *UAI '94*, pp. 102–109. 1994.

[2] W. Buntine. Operations for learning with graphical models. *J. of Artificial Intelligence Research*, 2:159–225, 1994.

[3] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: a Bayesian classification system. In *ML '88*. 1988.

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

[5] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[6] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *ML '95*. 1995.

[7] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI '93*, pp. 1022–1027, 1993.

[8] N. Friedman and M. Goldszmidt. Building classifiers using Bayesian networks. In *AAAI '96*. 1996.

[9] D. Heckerman. A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[10] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowlege and statistical data. *Machine Learning*, 20:197–243, 1995.

[11] D. Heckermann and D. Geiger. Learning bayesian networks: a unification for discrete and gaussian domains. In *UAI '95*, pp. 274–284. 1995.

[12] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *UAI '95*, pp. 338–345. 1995.

[13] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pp. 740–743. IEEE Computer Society Press, 1994.

[14] W. Lam and F. Bacchus. Learning Bayesian belief networks. An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[15] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[16] R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[18] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.