General Bayesian Networks and Asymmetric Languages

Joshua W. Comley
Department of Computer Science and Software Engineering
Monash University
Clayton, Victoria 3800, Australia
joshc@bruce.csse.monash.edu.au

David L. Dowe
Department of Computer Science and Software Engineering
Monash University
Clayton, Victoria 3800, Australia
dld@bruce.csse.monash.edu.au

Abstract

In this paper we introduce a new class of Bayesian networks. Unlike most Bayesian networks, it is capable of handling combinations of discrete and continuous attributes without the need to quantize continuous data. We achieve this by using a general class of decision tree to provide the conditional probability distributions, which has the added advantage of compactly representing context-specific independence.

Our main contribution, however, is the identification of a new family of joint distribution models that we label 'asymmetric' Bayesian networks. The nature of the joint distributions able to be expressed by these networks changes with regard to the total ordering of the nodes. We show that our proposed decision-tree-based Bayesian network class belongs to this family, and discuss other members.

We present an efficient algorithm for determining the best 'asymmetric' network structure, and suggest a modification for 'supervised' learning of network structure, which we argue should yield superior classification results.

Lastly, we compare our new classes of 'asymmetric' networks with two decision tree tools on 'real-world' classification problems. This produces encouraging results, with the 'asymmetric' networks yielding superior average classification accuracy.

1 Introduction

Bayesian networks (also referred to as belief networks) are a popular tool in artificial intelligence and statistical modelling. Their appeal is due largely to the intuitive graphical representation of the interdependencies between variables, and to the savings in computation afforded by conditional independence assumptions.

Bayesian networks provide a joint probability distribution over a set of attributes, $X = \{X_1, X_2, \ldots, X_n\}$, and express this as the product of n conditional probability distributions (CPDs). They are directed acyclic graphs (DAGs), with one node corresponding to each attribute. Each node provides a probability distribution over the values of the attribute it represents, conditional upon the values of the attributes represented by its parents. It is helpful at this point to introduce some nomenclature. We will use X_i to refer to both the attribute X_i and the node representing that attribute, as these are equivalent for the purposes of this paper. The set of parent nodes (or attributes) of X_i will be denoted P_i and, similarly, the set of direct descendants (or children) of X_i will be denoted C_i. To refer to a particular value taken by attribute X_i, we will use x_i. Likewise, p_i will refer to an 'instantiation', or configuration, of X_i's set of parent attributes P_i. Figure 1 shows an example Bayesian network structure (without depicting the probability distributions). For a general introduction to Bayesian network theory, see (e.g.) [23, chapter 15, section 5].

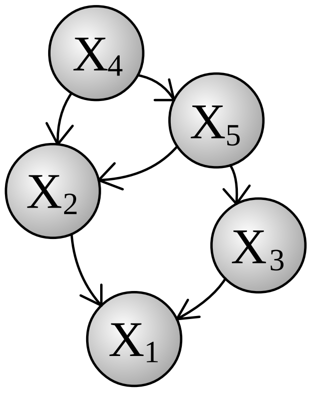

Figure 1: An example Bayesian network structure. Here $P_2 = \{X_4, X_5\}$, $C_2 = \{X_1\}$, $P_4 = \{\}$, $C_4 = \{X_2, X_5\}$, $P_1 = \{X_2, X_3\}$ and $C_1 = \{\}$.

Normally, Bayesian networks are concerned only with discrete (categorical) attributes, and express the conditional distribution $pr(X_i|P_i)$ as a conditional probability table (CPT) - with one row for each possible p_i, and one column for each possible x_i. The probabilities for any row should sum to 1. CPTs are easy to implement and interpret, and certainly quick to 'learn' - i.e. estimate their parameters, given data - but they suffer from some drawbacks. If X_i has many parent attributes then the table can quickly become unwieldy, especially if some of the attributes can take many values (this is because the number of configurations of P_i increases combinatorially with its cardinality). The awkwardness of such a large table is not so much of a problem in itself, but estimating reasonable parameters from anything but very large samples does become difficult. Large tables can be sparse, often with few (or no) data observed for some p_is. In addition, it may be that many rows of the CPT behave similarly. Stating different parameters for each row is often redundant, and we would like to join these rows together. This is known as 'context-specific independance' and has been addressed by (e.g.) Boutilier et al. in [4] by replacing CPTs with decision trees, mapping each p_i to a leaf node giving a distribution over X_i. This paper also uses decision trees in preference to CPTs, although our primary motivation is a little different, and our decision trees are somewhat more general.

As mentioned above, most Bayesian network tools are restricted to discrete data. Thus any continuous attributes in a data set are typically quantized, and treated as categorical. This is often acceptable, but has the unfortunate side effect of losing the notion of 'order'. For example, consider a continuous attribute 'height' that is quantized into four 'bins': '< 170', '170 - 180', '180 - 190', and '> 190'. A CPT has no concept that the value '> 190' is closer to '180-190' than it is to, say, '< 170'. In some domains, the notion of order is important and we do not wish to be at the mercy of our quantization algorithm. Some Bayesian network classes do not require discretization of continuous values. These networks typically only model continuous attributes, giving X_i as a linear combination of the attributes in P_i, plus a Gaussian noise term (see, e.g., [35]).

This paper presents a new class of Bayesian networks, able to handle a combination of discrete and continuous attributes, without the need for quantization.

Section 2 describes the principle of Minimum Message Length (MML) inference, which we use to learn both the structure of the network, and the decision trees (described in section 3) which provide the conditional probability distributions.

In section 4 we show how the decision trees can be used to construct the new class of Bayesian network. Unlike other Bayesian network models, the family of joint distributions able to be represented by this network class depends on the node ordering chosen. Section 5 examines how this leads to a natural and efficient method for inferring network structure.

Section 7 explores an alternative MML coding scheme for scoring networks where we know in advance which attribute we wish to predict. This method focuses on learning an accurate conditional distribution of our target attribute, rather than trying to learn the best joint distribution over all the attributes.

In section 8, we present encouraging results from several real-world classification problems, showing our new classes of Bayesian network out-performing various decision tree tools. We discuss our contribution in section 10.

2 Minimum Message Length

Minimum Message Length [29, 34, 32] is a Bayesian method of point estimation, based on Shannon's mathematical theory of communication [24]. When estimating parameters describing a body of observed data, MML constructs a two-part message. The first part encodes the parameters using a prior distribution, and the second part encodes the observed data using the distribution implied by the parameters. We choose the parameter estimates corresponding to the shortest overall message, which effectively maximizes the posterior probability[32, 33].

In its most general form, it is related to Kolmogorov complexity and can be used to infer any computable function [32]. It is invariant under 1-to-1 parameter transformations [30, 34, 28, 32], has general statistical consistency properties [1, 28][34, p241] and generally comes close to minimizing the expected Kullback-Leibler distance [9]. For comparisons between MML, the Minimum Description Length (MDL) works of Rissanen [20, 21, 22] and the works of Solomonoff [25, 26], Kolmogorov [13] and Chaitin [5], see Wallace and Dowe [32] and other articles [8, 6, 22, 26] in that special issue of the *Computer Journal.*

3 General Decision Trees

Decision trees are common tools in machine learning and data mining. They are easily interpreted by humans, and hence often preferred over less intuitive models. A decision tree performs tests on explanatory (input) attributes, and provides a model of a target (output) attribute - the parameters of which depend on the outcome of the tests.

This section describes a class of general decision trees that can model either discrete (categorical) or continuous target attributes, and handle either discrete or continuous input attributes. If the target attribute is discrete, each leaf will correspond to a different multi-state distribution. For continuous attributes, Gaussian density functions are used in the leaves. A branch node can test an n-valued discrete input attribute X_i with a simple test, assigning each possible value x_i to a separate sub-tree. A continuous input attribute X_j can be tested at a branch node using a 'hard' cut-point c. Any data item where $x_j < c$ will be assigned to the left-hand sub-tree, and any data item with $x_j \geq c$ will be assigned to the right-hand sub-tree.

3.1 MML Cost of a Decision Tree

The decision tree coding scheme used in this paper is an adaptation of that proposed in [36] - which in turn re-visited a scheme in [19].

In our MML framework we are attempting to minimize the length of encoding a decision tree and the (target) data it models. We may assume that the input data is already known by the decoder, as transmitting it will simply add the same constant term to the message length of any tree considered. Our message, then, will firstly state the tree. Once the receiver has decoded the tree she may use it, together with the input data, to derive a probability distribution over the target data. This allows her to decode the second part of the message, which transmits the target data using an optimal code based on this distribution.

The tree is transmitted in a depth-first, left-to-right fashion. For each node we firstly need to state whether it is a branch or a leaf. At the root of the tree this costs $\log_2(n+1) - \log_2(n)$ bits for a branch node, and $\log_2(n+1)$ bits for a leaf node, where n is the number of input attributes. For any non-root node, the codeword for a branch requires $\log_2 s$ bits, and a leaf requires $-\log_2\big((s-1)/s\big)$ bits. Here s represents the number of children of the node's parent, and hence is always greater than 1.

Following each 'branch' codeword, we need to state which of the input attributes will be tested. This requires $\log_2 A$ bits, where A is the number of attributes able to be tested at that branch[1]. If the input attribute to be tested is continuous, we also need to state the cut-point to be used. To do this, we borrow a scheme developed by Wallace [36] and used in both the seftware associated with that paper [36] and in [14, section 4.1]. We first order the observed values of our input attribute. We do not need to state the cut-point to any great precision - we only need to state which two values it lies between. We assign probability 1/2 (code-length 1 bit) to the cut-point being at the median observed value. Either quartile costs 3 bits (probability 1/8), octiles cost 5 bits (probability 1/32), and so on. (This coding scheme uses that fact that $\frac{1}{2} + \frac{2}{8} + \frac{4}{32} + \ldots = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 1$.) It is thus very cheap to do a 'rough' split, sending half of the values either way. Each half of these values now has a new median, etc., and may be partitioned again by subsequent branch nodes.

Following each leaf codeword, we state the parameters associated with the leaf - either μ and σ^2 for a Gaussian distribution if the target attribute is continuous, or $pr(v_1), \ldots, pr(v_{k-1})$ for a k-state distribution if the target attribute is discrete and can take the values $v_1, \ldots, v_k$.

[1]A discrete attribute cannot be tested by more than one branch node in any path. If a branch tests a discrete attribute, A is decremented for each of the child trees.

[29] and [3] give well-behaved priors and coding schemes for both the Gaussian and multi-state models respectively (see also [33]).

3.2 Searching for a Decision Tree

Given a set of training data, we use a simple 'lookahead-0' greedy search algorithm to learn a decision tree. We begin by setting the root node to be a leaf, and optimising the parameters for the model it contains. This forms our 'null hypothesis', where the values of the input attributes have no bearing on the distribution over the target attribute.

Now we consider whether changing the leaf to a branch testing attribute X_i would improve the cost (i.e. the message length) of our tree. We assume the children of this potential branch are leaves, and optimize their parameters according to the training data they pertain to. We try this for each input attribute X_i in turn. If none of the potential branches have a cheaper cost than the leaf node, we terminate the search. Otherwise we change the leaf node into the cheapest branch considered, and repeat the algorithm for each of the resulting leaves. The lookahead-0 search suffers from several drawbacks, most notably the difficulty discovering 'XOR' relationships. Despite this, its simplicity and efficiency make it a popular and effective algorithm.

4 'Asymmetric' Bayesian Network Classes

In this section we construct a class of Bayesian networks using the decision trees described in section 3. Each node X_i of the network contains a decision tree modelling X_i as a function of its parent attributes, P_i. Nodes with no parents use a tree which is just a leaf - as the distribution is not conditional on any input data. If X_i is discrete, a tree with multi-state leaf distributions will be used - otherwise the leaves will contain Gaussian density functions, as described in section 3. As usual the joint distribution $pr(X)$ is expressed as the product of the conditional distributions, $\prod_{i=1}^{n} pr(X_i|P_i)$.

The interesting thing here, however, is that by changing the order of the nodes we actually alter the family of joint distributions able to be expressed. In this paper we name such classes of network 'asymmetric Bayesian networks' due to the asymmetry of the joint distribution with respect to node order.

4.1 A Decision Tree Example

To illustrate this asymmetry, let us consider a simple example with only two attributes - a continuous attribute X_c and a binary attribute X_b. There are two possible node orderings - (X_b, X_c), and (X_c, X_b).

Let us first examine the node ordering (X_b, X_c). This network models the joint distribution, $pr(X)$, as

$$pr(X) = pr(X_b).pr(X_c|X_b)$$

where $pr(X_b)$ is given by a simple two-state distribution, and $pr(X_c|X_b)$ is a decision tree, possibly testing X_b at a branch node, and modelling X_c in the leaves with Gaussian density functions.

Now let us look at the node ordering (X_c, X_b). This network models the joint distribution, $pr(X)$, as

$$pr(X) = pr(X_c).pr(X_b|X_c)$$

Here $pr(X_c)$ is given by a single Gaussian density function, and $pr(X_b|X_c)$ is a tree possibly partitioning X_c with hard cut-point tests at branch nodes, and offering a different two-state distribution over X_b in each leaf.

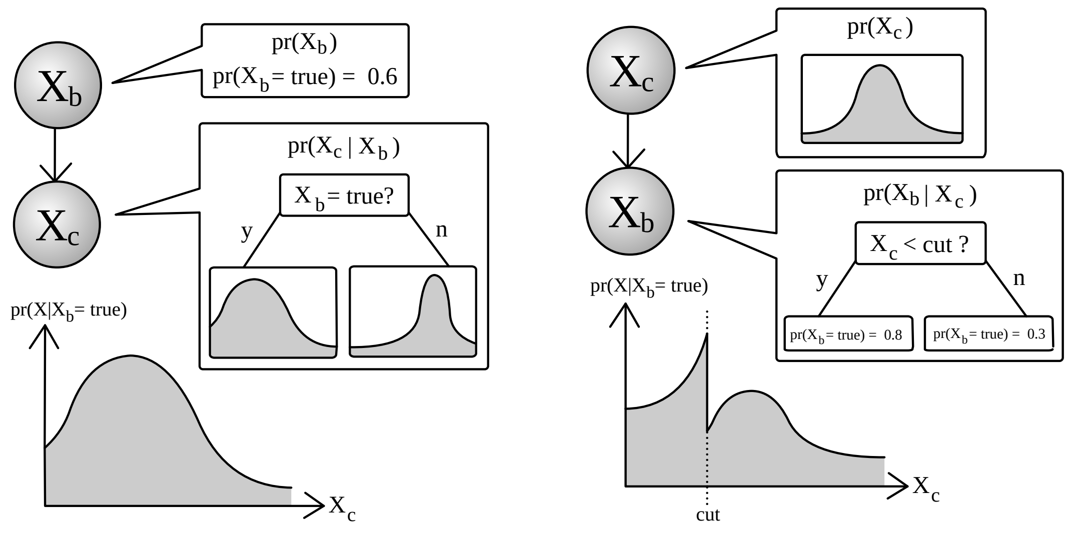

Figure 2: This figure shows the difference that node order can make to the nature of the joint distribution when dealing with asymmetric Bayesian networks. Two networks are depicted - one on the left with the ordering (X_b, X_c), and one on the right with the ordering (X_c, X_b). To the right of each node we depict the conditional probability distributions it contains. Below each network is a (rough) graph showing how, when $X_b = true$, $pr(X)$ varies with X_c. NOTE: This figure is not drawn accurately or to scale - it is intended only to give an idea of the behaviour of our class of asymmetric networks.

These two expressions of the joint distribution are clearly quite different in nature. We can see this by holding X_b constant (say $X_b = true$), and plotting the joint distribution $pr(X)$ as a function of X_c. This is illustrated in figure 2. In the first node ordering, (on the left in figure 2) $pr(X)$ varies smoothly with X_c (according to a Gaussian density function). In the second node ordering, (shown on the right in figure 2) $pr(X)$ will feature discontinuities at the cut-point(s) where X_c is partitioned by the branch node(s) of the decision tree giving $pr(X_b|X_c)$. In most cases, one node-ordering (or family of joint distributions) will be better suited to a data set than another. In this regard our class of networks contrasts strongly with the usual class of Bayesian networks using CPTs, in which any two node orderings can be made to represent the same joint distribution given appropriate CPT parameters.

4.2 A Univariate Polynomial Example

The decision trees used in this paper are not the only class of conditional probability distribution that gives rise to asymmetric networks. Nor is it necessary to have both continuous and discrete attributes for asymmetric networks to exist. Another example of a CPD language that leads to asymmetric networks is univariate polynomial functions with Gaussian noise. Here an attribute X_i with parent X_j is modelled as a probabilistic polynomial function[2] of the form:

$X_i = a_0 + a_1 x_j + a_2 x_j^2 + \ldots + a_d x_j^d + N(0, \sigma^2)$ where d is some maximum degree of polynomial able to be inferred.

[2]restricted to non-negative integer powers.

An attribute X_k with no parents could be modelled by $X_k = a_0 + N(0, \sigma^2) = N(a_0, \sigma^2)$, which is simply a Gaussian density function.

Imagine that a network with node ordering (X_j, X_i) defines X_j to be governed by the Gaussian distribution $N(\mu_j, \sigma_j^2)$, and X_i to come from the distribution $x_j^3 + N(\mu_i, \sigma_i^2)$. If we reverse the node order then there is no way, given our CPD language, to describe the same joint distribution. The inverse expression of the joint distribution would be that $X_j = (x_i - \mu_i + N(0, \sigma_i^2))^{\frac{1}{3}}$, and that X_i is governed by $(\mu_j + N(0, \sigma_j^2))^3 + \mu_i + N(0, \sigma_i^2)$, but this clearly does not fall within our language of (non-negative) integer power polynomials. There are no parameters we could choose for $pr(X_i)$ and $pr(X_j|X_i)$ that could capture this relationship.

Indeed, if this were the true distribution from which the data were drawn, we would do much better with the node ordering (X_j, X_i) than with ordering (X_i, X_j). It is even possible that if we chose the latter ordering, then the dependence of X_j on X_i would not be detected. It is this kind of concept that originally motivated this work, and earlier, related ideas by Dowe in [11], which we now outline briefly for historical reasons in Section 4.3 before proceeding to the determination of the network structure in Section 5. Section 4.3 includes additional examples (e.g., mixture modelling[11]) to the decision tree example of Section 4.1 and the univariate polynomial example of Section 4.2, but it is included primarily for historical reasons and has some slightly different terminology and notation. The reader can safely proceed to the determination of the network structure in Section 5 without loss of continuity.

4.3 The Historical Motivation for "Inverse Learning"

Dowe's original work[11] was concerned with one or two groups of attributes. A simple example of one group of attributes is mixture modelling[11], discussed again towards the end of this section. Let us proceed now to two attributes - or groups of attributes - A and B. Regardless of which of A and B was to be the target attribute(s), to be modelled in terms of the other (explanatory) attribute(s), he advocated seeking to find the best joint distribution of $A \& B$. This best joint distribution of $A \& B$ was to be found by choosing the cheapest of two MML messages - one transmitting A followed by $B|A$ and the other transmitting B followed by $A|B$. When A and B contain one attribute each, this relates closely to the simple case of Bayesian networks where we have only two nodes.

One original motivation for this was the inference of protein secondary structure from the protein's primary, amino acid, sequence. Inference of the secondary structures (SS) as a function of the amino acids (AA) (see, e.g., [10]) was not accounting for the high degree of serial correlation in the secondary structure sequence - e.g., helices are stable and potentially long structures with run lengths of at least 3. Using a language of only decision graphs [17, 16] and Markov models, Dowe suspected that modelling SS (using a Markov model to deal with the serial correlation) and then modelling AA as a decision graph function of SS could lead to a better fit to the data than modelling AA (which shows at most scant evidence of serial correlation) and then (as in [10]) modelling SS as a decision graph function of AA. Dowe suspected that this improved fit to the joint distribution of $AA \& SS$ would lead in turn to a better model of SS given AA. Given the interchangeability of code lengths (l) and probabilities (p) via the relation $l = -\log p$ (and, depending upon whether l is in bits or nits, either $p = 2^{-l}$ or $p = e^{-l}$) - see, e.g., [30] - the reasoning preceded along the following lines. Traditional supervised learning of the data using MML (e.g., [10]) encodes $H_{SS|AA}; SS|\{AA, H_{SS|AA}\}$. Encoding the entire data-set so that we can compare it with the inverse model entails prepending this code with a code for the amino acids, so we now encode $AA; H_{SS|AA}; SS|\{AA, H_{SS|AA}\}$, which, by Bayes's theorem, is equivalent to encoding $AA \& H_{SS|AA} \& SS$ or, in turn, $AA \& SS \& H_{SS|AA}$. Using the inverse model, we encode $SS; H_{AA|SS}; AA|\{SS, H_{AA|SS}\}$, which, again by Bayes's theorem, is equivalent to encoding $SS \& H_{AA|SS} \& AA$ or, in turn, $AA \& SS \& H_{AA|SS}$. A ventured criticism is that we already know the observed values of AA and are interested only in encoding a two-part message for SS, so why would we bother to encode AA? One response is that the inclusion or exclusion of AA from the message is equivalent to the addition or otherwise of the constant quantity of $-\log m(AA)$ from the message length, where $m(AA)$ is the marginal probability of AA having occurred given the relevant prior probabilities and likelihood functions. Another response is merely to appeal to the consistency[1] [34, p241][32] and invariance[30, 34, 32] of MML - it is explicitly shown in [32] how MML relates to Turing machines and gives an optimal two-part message encoding the entire data-set. Dowe at least initially contended that the *entire* data-set includes AA in addition to SS. Finally, with regards to new test data once an inverse/implicit model has been inferred, with AA fixed and a change in SS affecting the cost both of SS and of $SS|\{AA, H_{SS|AA}\}$, Dowe suggested simulated annealing as a viable means of search.

Before proceeding on to another use of Dowe's motivating examples, mixture modelling[11], we briefly describe Comley's contributions, and (our understanding of) Chris Wallace's modification[3]. Comley refined Dowe's original 'inverse learning' message length format, allowing it to fit more properly into the MML framework. This refinement was largely conceptual, but did address a (possibly small) under-statement of the message length in Dowe's scheme. (Interestingly, the most tractable approximations to Dowe's version would probably lead directly to Comley's refinement.)

Comley also developed a generalization of Dowe's original idea, allowing more than two (groups of) attributes. This drew attention to certain parallels with Bayesian network theory, and is the basis of the work presented in this paper. Additionally, Comley proposed several methods of searching for optimal network structure, one of which is presented in Section 5. The scheme presented in section 5.1, and referred to as 'unsup-net' in section 8, uses Comley's refined MML message.

In private communication, Chris Wallace suggested another modification to Dowe's scheme. As the authors understand, this involves focusing on a 'target' attribute. Whereas Dowe's original version had a message transmitting all the data and a hypothesis (model) for some of the attributes, and Comley's refinement has a message both modelling and transmitting *all* the data, in (our understanding of) Chris Wallace's modification, any message only has to transmit the target attribute. This is done either by using an explicit conditional model of the target attribute given the remaining attributes, or by transmitting such a distribution implicitly - in the form of a joint distribution over the target attribute and some other attributes, from which we may extract a 'cross-section'. This scheme is concerned with finding the best *conditional* distribution of the target attribute, given the remaining attributes, rather than the best joint distribution over all the attributes. It is discussed further in section 7.

We now conclude this section with two further historically motivating example for "inverse learning" (or "implicit learning"), the first of which is a case where we are primarily interested in one of several continuous-valued attributes. Suppose the true underlying distribution of the data is a mixture model and that our modelling languages include mixture modelling[15, 33, 31] and decision trees with 'hard' cuts on continuous-valued attributes (see Section 4.1 or [36]) but with (univariate or multivariate) Gaussian distributions rather than multinomial distributions in the leaves, akin to the leaves in the left part of Figure 2. If our true underlying model is of the form (X_c, X_b), such as the logistic model, then even if X_c is the target attribute, we do best to infer the underyling model rather than a model chosen merely because it is of the form (X_b, X_c) (such as the model at the left of Figure 2).

Because the true underlying data-generating distribution is a mixture model, as the amount of data increases, the consistency, invariance and other properties of MML (see, e.g., [1, 32] and section 2) will correctly lead us to infer a mixture model of the joint distribution. With

[3]from a personal communication

our target attributes as one of these attributes, inference of this mixture model leads us in turn to correctly give the cross-section of this mixture model as the most viable model for our attribute of interest - rather than, instead, some approximating decision tree with hard cuts and a univariate Gaussian distribution in each leaf.

Briefly, another historically motivating example is a further case with the two attributes - one discrete (binary) and one continuous - as in Section 4.1, but where we consider an additional (third) model in which X_b is dependent on X_c, and the probability of one of the binary classes is a logistic function of the continuous-valued attribute.

Dowe originally named this concept 'inverse learning', later referring to it as 'implicit learning'. A further account of the ideas in this paper and the contributions of both Comley and Dowe is planned to appear in [7].

Having given Dowe's original historical ideas and motivation, including the mixture modelling example[11], we now proceed to the determination of network structure.

5 Determining the Network Structure

In a traditional Bayesian Network (BN), altering the node ordering will have no impact on the family of joint distributions able to be expressed[4]. Network structure is generally decided by domain experts, or based on some notion of minimising the node connectivity (without causing too much damage to the expressiveness of the network). The aim is usually to achieve some trade-off between a highly connected, computationally expensive network - with large CPTs that can capture complex interdependencies - and a simpler, sparse network that is easy to understand and can perform computations more cheaply but that may not do a good job of capturing all the relationships present in the data.

When using the class of networks described in section 4 different total node orderings correspond to different families of joint distribution. This means that of two competing networks with the same connectivity between attributes but with different node orderings, one is able to give a better model of the joint distribution that the other.

We now describe the first author's idea of how this can be used to develop an efficient and automatic search for the best network structure, using the class of decision trees described in section 3 as our CPD language. Consider two nodes X_i and X_j, in an asymmetric network. It may be that X_i can be nicely described as a probabilistic function of X_j, but when X_j is modelled as a probabilistic function of X_i, no dependence can be found (see section 4.2 for an example). We can conclude from this that there is some merit in placing X_i after X_j in the total ordering, but that placing X_i before X_j will not help us. We use this concept in the strategy below to help us place some constraints on the partial ordering of the attributes. The strategy works in several steps.

1: **Learn the initial decsion trees.**
For each attribute $X_i \in X$, learn a tree modelling X_i, with $X \setminus \{X_i\}$ as input.

2: **Establish a list of useful directed links.**
Begin with an empty list L of useful directed links. For each attribute $X_i \in X$, check whether each other attribute X_j $(j \neq i)$ is tested by the decision tree modelling X_i. If the tree modelling X_i *does* test X_j, then add a directed link $X_j \rightarrow X_i$ to L. This indicates that placing X_j before X_i in the total ordering may be useful.

3: **Score the total orderings.**
We use a stochastic process to investigate various total orderings[5]. We choose an initial

[4]providing that no connections between attributes are removed.
[5]When n (the number of attributes) is small, it is feasible to exhaustively check all $n!$ total orderings.

total ordering, and repeatedly permute it by swapping the positions of two adjacent attributes. A fixed number of permutations is performed (that increases with the number of attributes, n). Each total ordering considered is assigned a score, equal to the number of useful directed links in L that it exhibits - i.e., if X_i occurs before X_j in a particular total ordering, then that ordering exhibits $X_i \to X_j$, but not $X_j \to X_i$. Note that at this stage we have only had to learn the initial n decision trees from step 1. Examining total orderings does not require additional CPDs to be learnt, making this step quite fast. Throughout the pseudo-random process, we keep only those orderings with the (equal) maximum score.

4: **Remove 'useless' links.**
We now iterate through each of the total orderings with the equal maximum score, treating each of them as a fully-connected network (currently without CPDs in the nodes). We remove all directed links that do not feature in L.

5: **Calculate structural equivalency.**
After step 4 we are left with a list of total orderings representing partially connected network structures. Many of the structures represented will be equivalent[6], and should be treated as a single structure. We group the total orderings by structural equivalence, and choose an arbitrary member from each group as a representative. We record the cardinality of each group.

6: **Learn all required decision trees.**
We now build a list, DT, of all the decision trees required to give us CPDs for each representative network. For each network, we examine all its directed links. A link $X_j \to X_i$ means that X_j is one of X_i's parents, and should be included in the set of input attributes for the tree modelling X_i. For example, one network may require (amongst other trees) a decision tree modelling X_2 given $\{X_1, X_3, X_4\}$ while another network may require the decision tree modelling X_2 to have $\{X_1, X_5\}$ as input attributes. Both these trees would be learnt, and added to the list DT. A large saving can usually be made here, as any one decision tree is often used by many networks, and only needs to be learnt once.

7: **Cost the representative networks with MML.**
We now have a list of candidate networks (each representing a group of networks with equivalent structure), with decision tree CPDs in the nodes. We cost each network using the MML scheme outlined below in Section 5.1, and choose the network with the lowest cost.

5.1 A Minimum Message Length Scheme for Bayesian Networks

This section briefly describes how one could derive a message length for a group of structurally equivalent networks. This coding scheme is specifically designed to be used with the search in Section 5, as it makes assumptions that take advantage of our candidate networks having an equal degree of connectivity.

We will transmit the attributes in the order dictated by the representative network's total node ordering. Firstly, though, as we are comparing networks with different orderings, we will need to let the receiver know what the ordering is. There are $n!$ possible total node orderings, where n is the number of attributes in our data set. So encoding any one of them (using a uniform prior) requires $\log_2(n!)$ bits. But remember that any two networks in the same group

[6]Two networks are structurally equivalent if they exhibit an identical collection of directed links.

are actually equivalent models (and have different total node orderings). We should derive a codeword for each *distinct* model - otherwise equivalent networks would be 'stealing' from each other's code-spaces, making the model they represent seem less probable. If there are k networks in the group, then the group's codeword will be of length $\log_2(n!) - \log_2(k)$ bits.

Now that the receiver knows the total ordering of the network, she can establish the connectivity, and determine which CPDs are to be sent. Here we assume that she already knows the list of 'useful' directed links, L. This is a valid assumption, as this is common information for all candidate networks. We begin with no connections between nodes. Then for each link $X_j \rightarrow X_i$ in L, we insert it into the network if X_j occurs before X_i in the total ordering.

We now transmit each decision tree and the data it models (see section 3.1) according to the total ordering. For example, in the network shown in figure 1, we would transmit:

1a: The tree giving $pr(X_4)$,

1b: The values of X_4, using the tree in 1a,

2a: The tree giving $pr(X_5|X_4)$,

2b: The values of X_5, using the tree in 2a. Because we have already transmitted the values of X_4, the receiver will know which leaf of the tree (in 2a) to use for each item.

3a: The tree giving $pr(X_2|X_4, X_5)$,

3b: The values of X_2, using the tree in 3a. Etc. ...

Our total message length for a group of k equivalent networks is then simply $\log_2(n!) - log_2(k)$, plus the cost of transmitting each decision tree and the relevant data, as given by section 3.1.

Note that this scheme does not penalize groups of highly connected networks in the transmission of the network structure. All network groups considered have the same degree of connectivity, because they all exhibit the same (maximum possible) number of directed links in L. Any more sparse networks would have been discarded in step 3 of the search algorithm above. It is also worth noting that often not all links in L can be exhibited by any given network - consider the case where L contains the links $X_i \rightarrow X_j, X_j \rightarrow X_k$, and $X_k \rightarrow X_i$. Because Bayesian networks are directed *acyclic* graphs, any network could only exhibit two of these links.

5.2 Potential Problems with the Search Algorithm

Recall that when performing step 2 - i.e., establishing which directed links are useful, and which should be removed - we assert that a link $X_j \rightarrow X_i$ is useful only if the decision tree modelling X_i (from step 1) tests X_j at a branch. While this generally works well, there are two problems we may face. These are outlined below.

5.2.1 Falsely Detecting Dependencies

Imagine that we have learnt a tree in step 1 where the target attribute X_i depends on a Boolean attribute X_b, and, *if X_b is true*, also depends on X_j. Because the tree tests both X_b and X_j, we conclude that the links $X_b \rightarrow X_i$ and $X_j \rightarrow X_i$ are worthwhile. Imagine now that we go with the ordering X_j, X_i, X_b. Now the link $X_j \rightarrow X_i$ may be useless - we may not be able to detect any dependency of X_i on X_j without the presence of X_b. It would be better to have removed this link, but it is too late because the structure (and connectivity) is decided before the tree giving $pr(X_i|X_j)$ is inferred, and it is only when we infer this tree that we discover it does not actually test X_j.

5.2.2 Failing to Detect a Dependency

If some attributes are highly correlated, they may 'over-shadow' each other. e.g. X_i has a strong dependence on X_j, and a weaker (but still important) dependence on X_k. The decision tree giving $pr(X_i|X_j, X_k)$ tests X_j at the root node, nicely partitioning the classes. Each leaf now decides not to bother testing X_k, due to fragmentation of data, and the minimal extra purity gained. So we conclude that X_i does not depend on X_k - but in fact this independence is conditional on X_j being present as a parent attribute. Imagine an ordering X_k, X_i, X_j where we would decide to remove the link $X_k \rightarrow X_i$. Now our encoding of X_i will not benefit from any correlations.

Another cause of this error is that in the presence of many input attributes, stating which attribute is to be tested at any branch becomes expensive (see section 3.1). A 'border-line' branch may be rejected on this basis whereas in the actual network (where there are fewer input attributes) it will be cheaper to state that branch and it may be accepted.

6 Using a Network for Prediction

This section briefly explains how to extract a conditional probability distribution $pr(X_i|X \setminus \{X_i\})$ from a Bayesian network representing the joint probability distribution $pr(X)$.

If X_i is the last node in the network (or has no children), then we can simply use the CPD $pr(X_i|P_i)$ because, given P_i, X_i is inpedendent of all other attributes. For the class of networks presented in this paper, this means we simply feed our value p_i into the decision tree modelling X_i, and read our CPD from the relevant leaf.

If the node X_i has children, however, the situation is a little more complex. Here we use Bayes's rule (hence the term Bayesian network) to re-write $pr(X_i|X \setminus \{X_i\})$ as follows:

$$pr\big(X_i = x_i\big|(X \setminus \{X_i\} = x \setminus \{X_i\})\big) = \frac{pr(X_i = x_i|P_i = p_i).pr(C_i = c_i|X_i = x_i, P_i = p_i)}{pr(C_i = c_i|P_i = p_i)} \qquad (1)$$

where $x \setminus \{X_i\}$ is meant to represent a particular value taken by $X \setminus \{X_i\}$.

If the goal is to simply predict the most probable value for X_i, given values for $X \setminus \{X_i\}$, then we can ignore the denominator $pr(C_i = c_i|P_i = p_i)$ as this simply serves as a normalization factor. If we do want actual probabilities, however, we must approximate $pr(C_i = c_i|P_i = p_i)$, as the network does not give us this conditional probability directly. We may approximate it as follows:

$$pr(C_i = c_i|P_i = p_i) \approx \frac{pr(C_i = c_i|X_i = x_i, P_i = p_i)}{\sum_{x'_i \in X^*_i} \Big(pr(X_i = x'_i|P_i = p_i).pr(C_i = c_i|X_i = x'_i, P_i = p_i)\Big)} \qquad (2)$$

where X^*_i is the space of all possible values for attribute X_i.

Used like this, extracting $pr(X_i|X \setminus \{X_i\})$ from a Bayesian network can be seen analogously to Bayesian inference, where our prior on 'hypothesis' x_i is given by the CPD in node X_i, and our conditional probability (or likelihood) of observing 'data' c_i, given 'hypothesis' x_i, is the product of the conditional probabilities given by each node in C_i.

7 'Supervised' Networks

We present in this section an alternative to the MML costing scheme given in 5.1. This is the authors' understanding of an idea conveyed by Chris Wallace in private communication. The alternative scheme can be used when we know, before inferring the network, which attribute it is that we wish to predict. This is often the case in practical classification situations, where there is usually a particular attribute of interest which is difficult to measure, that we want to predict based on the rest of the (more easily) observed attributes. In this section we will refer to such an attribute as the 'target' attribute, and label it as X_t.

This scheme focusses on learning an accurate *conditional* distribution of X_t given $X \setminus \{X_t\}$ - as opposed to learning an accurate joint distribution over all of X. Wettig et al. [37] refer to networks that result from such schemes as 'supervised' networks, and to networks that have attempted instead to optimise the joint distribution as 'unsupervised' networks. We adopt this terminology, as it draws attention to the role of networks and their distributions in classification tasks.

For the asymmetric networks presented here, the structure, connectivity and parameters required to represent the best joint distribution[7] may differ significantly from those required to represent the best conditional distribution[8] of X_t. We expect, when the task is to predict/classify X_t, that the supervised network will produce better results.

Our proposed MML scheme for supervised networks differs only sightly to that for unsupervised networks presented in section 5.1. The major difference is that the supervised scheme assumes that the values for $X \setminus \{X_t\}$ are common knowledge, and need not be included in the message. We transmit the network structure and the decision tree parameters in exactly the same manner. In the supervised scheme, though, we do *not* transmit the data. After she has decoded the network the receiver may use it, together with the values for $X \setminus \{X_t\}$, to derive a conditional probability distribution $pr(X_t|X \setminus \{X_t\})$ using the method explained in section 6. It is using this distribution that the values of our target attribute, X_t, are transmitted.

This scheme favours networks that provide good compression of X_t (by offering accurate distributions over $X_t|X \setminus \{X_t\}$) - as opposed to the unsupervised scheme (presented earlier in this paper) which places equal importance on the compression of each attribute. This scheme is referred to as 'sup-net' in Section 8 - an abbreviation of 'supervised-network'.

8 A Comparison of Classifiers

In this section we compare the performance of four classifiers on three real-world data sets. The classifiers are:

- **MML-DT:** This is a decision tree tool that infers models from the class of decision trees described in section 3. It uses an MML costing metric (see section 3.1) similar to that in [36] and a lookahead-0 greedy search algorithm (see section 3.2). This method is equivalent to a supervised network where $P_t = X \setminus \{X_t\}$.
- **C5:** C5 [18] (and its fore-runner, C4.5) are popular decision tree tools used for classification. C5 does not use the MML principle and is widely used as a performance benchmark in classification problems.
- **unsup-net:** This is the algorithm presented in this paper for learning unsupervised asymmetric Bayesian networks.
- **sup-net:** This is the modified algorithm (see section 7) that learns supervised asymmetric Bayesian Networks.

[7]i.e. for an unsupervised network,
[8]i.e. for a supervised network

Data-set	Target Attribute	Other Attributes
Iris	3-valued discrete	4 continuous attributes
Wine	3-valued discrete	12 continuous attributes
Ecoli	8-valued discrete	7 continuous attributes

Table 1: A summary of the nature of the three data-sets *iris*, *wine* and *ecoli*.

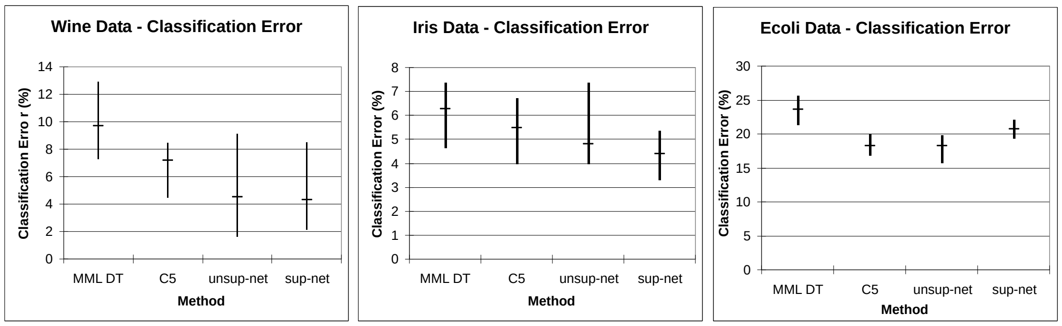

Figure 3: Best, worst and average performance for each of the methods described in section 8.

8.1 Experimental Setup

Tests were performed on three data-sets - *iris*, *wine*, and *ecoli* - which are all available from the UCI machine learning repository [2]. All data-sets feature a discrete target attribute (class), which allowed us to meaningfully compare our results with C5, because although our asymmetric networks can just as easily handle continuous targets, C5 and many other classifiers cannot. Table 1 summarizes the nature of each data-set.

Each data set was partitioned using 10-fold cross-validation, creating 10 different training-testing pairs. For each pair, the training set comprised 90% of the original data, and the testing set comprised the complementary 10%. The 10 testing sets were mutually exclusive. The same pseudo-random number seed was used to partition the data for each method, ensuring all four methods were working with exactly the same data.

Each method learnt a model from each of the 10 training sets, and used it to predict the target attribute in the corresponding test set. Methods were scored using their percentage classification error, averaged over the 10 data sets.

This entire procedure was repeated 10 times, using different seeds to partition the data each time (this gave us a total of 100 different training-testing data sets for each of *iris*, *wine*, and *ecoli*). For each method, we recorded its best, worst and average score over the 10 repetitions. This was done because the results from a single (10-fold) cross-validation run can be misleading, especially if the data was partitioned unusually.

8.2 Results

Figure 3 shows - for each database - the best, worse, and average performance of each method over the ten 10-fold cross-validation experiments. To clarify, take the performance of unsup-net on the wine database - its best score is just under 2%. This means that in one of the cross-validation experiments, the average classification error over the 10 training-testing pairs was just under 2%.

8.3 Discussion

The tests performed in this paper are quite robust and produce encouraging results. Both the supervised and unsupervised classes of network can be seen, on average, to out-perform C5 and the MML decision tree classifier for the *wine* and *iris* data sets. This indicates that there is some merit in tackling classification problems with asymmetric Bayesian networks, as this allows us to model families of conditional distributions of $X_t|X \setminus \{X_t\}$ that are not able to be expressed by simpler, explicit models such as decision trees.

As expected, we see slightly better performance (on the *wine* and *iris* data) from the supervised network than from the unsupervised one.

The ecoli data set produces some unexpected results, and while the unsupervised network performs marginally better than C5, the supervised network seems to have more difficulty - although it still easily out-performs the MML decision tree that models X_t explicitly. We feel that this result warrants more research into the mechanisms of prediction for supervised vs. unsupervised networks, with the aims of establishing what features of a data set cause difficulty, and perhaps learning how to overcome this.

9 Probabilistic Scores and Issues for Further Research

The conditional probabilities provided by our network classes were, on the most part, significantly better than those offered by the decision tree classifiers. However there were several test cases - in the wine and ecoli data sets - to which the inferred networks assigned very low probabilities. While this did not significantly impact on the total classification error, it had a notable effect on the network's probabilistic score (see, e.g., [12, 27], [14, Table 2]). These cases bring attention to some interesting issues.

The issues occur when our network contains a decision tree modelling a continuous attribute, X_c, (with Gaussian distributions), and testing the (discrete) target attribute, X_t, at a branch node.

The problem is when in one of the leaves the estimate of the variance of X_c is too small. In our test case we know the value $X_c = x_c$, and we wish to assign probabilities to each possible value of X_t. Different choices of x_t will lead to different leaves in our tree, and hence to different Gaussian distributions over X_c. If, in any of these leaves, the estimated mean of X_c differs much from the observed x_c, and the estimated variance of X_c is very small, then the corresponding x_t will be assigned an extremely low probability. Under the probabilistic scoring system, where a method is penalized with the negative log-likelihood of the test data, this can present a significant problem.

So when is the estimate of the variance of X_c too small? We have identified three related conditions where the estimation of the variance 'breaks down'.

The first condition is when all the training data in a leaf node has exactly the same (continuous) value v. When we fit a Gaussian density function to this data, we choose $\mu = v$, but the estimation of the variance, σ^2, poses a problem. Blindly using traditional formulae will yield $\sigma^2 = 0$, which will clearly have disastrous consequences for probabilistic score. It is helpful here to consider that any continuous measurement is only ever made to a finite accuracy ($\pm\epsilon$). For the trials in this paper we assume that the true value of each data item is equally likely to be anywhere in the range $[v - \frac{\epsilon}{2}, v + \frac{\epsilon}{2}]$. Our expectation of σ^2, then, works out to be $\epsilon^2/12$ (see [31, Sec. 2.1]).

While $\sigma^2 = \epsilon^2/12$ may be a reasonable estimate given a large number of data, it begins to seem a little bold for smaller training samples. Consider if we had only 2 data (with identical values) from which to infer our Gaussian distribution - such a small estimate of σ^2 hardly seems reasonable. Continuing this line of thought brings us to the second condition; fitting a

Gaussian distribution to a single data point. Again, estimating μ is not a problem. But here we would intuitively like a very large σ^2.

The third condition seems even more bizarre; how to estimate μ and σ^2 from no data? Fortunately, we are using MML which is a Bayesian method of point estimation, and as such acknowledges that we have some prior knowledge or belief of the nature of μ and σ^2. This can help us make some reasonable decisions - for instance, if we believe that μ has uniform probability of lying in the range $[a, b]$, then we can choose an estimate $\hat{\mu} = \frac{a+b}{2}$.

An alternative approach to this situation is to estimate the Gaussian distribution from *all* the training data - i.e., in the absence of any data we could borrow the data from the other leaves, giving us a 'prior' belief (of sorts), on how we expect data in our 'empty' leaf to behave.

The last two conditions can be seen in some sense to be special cases of the first - i.e. when all (continuous) training data has the same value. Intuitively, we feel that as the number of data decreases, it is less reasonable to estimate a small σ^2. Principles of Bayesian averaging also suggest that increasing σ^2 would improve the probabilistic score. Even though these conditions can easily be detected, and addressed using 'ad-hoc' techniques to increase the estimate of σ^2, the authors feel that these conditions warrant further research, to find a principled MML-based approach.

It should be pointed out that in most research these issues are avoided altogether - and it is not surprising as at first glance they seem like rather absurd special cases, and unlikely to be met in practice. But this is not the case. Consider a decision tree modelling a continuous attribute, and testing a discrete attribute with (say) 3 possible values. We will have one leaf for each of the 3 possible outcomes of this test. Now imagine that in our training set one of these values occurs only once. While two of the leaves will have plenty of data from which to infer a Gaussian distribution, the other leaf will have only a single data point.

10 Conclusions and Future Directions

We have presented a general class of Bayesian network that uses decision trees as conditional probability distributions. It can efficiently express context-specific independence, and is capable of modelling a combination of discrete and continuous attributes. We have suggested that when we know which attribute is to be predicted, it may be better to use a 'supervised' network rather than an 'unsupervised' one. We have proposed a modification to our algorithm to allow for this.

Our main contribution, other than extending Bayesian networks to handle continuous *and* discrete data, is the identification of 'asymmetric' networks, and the proposal of an efficient scheme to search for node order and connectivity. We have shown our class of networks to perform favourably on real-world classification tasks.

There are several interesting directions for future work. The unexpectedly poor behaviour of the supervised network on the ecoli data set warrants a more thorough analysis of the situations in which we can expect the supervised approach to be beneficial.

The conditional probabilities provided by our networks were notably better than those offered by the decision tree classifiers on almost all the test data. However the wine and ecoli data sets contained several cases to which the inferred networks assigned very low probabilities. This phenomenon is explored in section 9, where we have identified several areas for further research.

Finally, we feel it would be interesting to investigate other classes of asymmetric network using different languages for the conditional probability distributions, such as the family of univariate polynomials presented in section 4.2.

11 Acknowledgments

The authors would like to acknowledge Chris Wallace, Lloyd Allison and Trevor Dix for discussions over the years, dating back to Dowe's original presentation on 25th June 1996. We explicitly thank Chris Wallace for explaining to the best of our understanding his modification (see sections 7 and 4.3) which we have interpreted as the sup-net scheme in this paper.

David Dowe would like to take the opportunity in this work to especially thank his mother, his family, his friends and all those who have been helpful, supportive and encouraging in his career.

References

[1] A. R. Barron and T. M. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, 37:1034–1054, 1991.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[3] D. M. Boulton and C. S. Wallace. The information content of a multistate distribution. *Journal of Theoretical Biology*, 23:269–278, 1969.

[4] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfeth Conference (UAI-1996)*, pages 115–123, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[5] G. J. Chaitin. On the length of programs for computing finite sequences. *Journal of the Association for Computing Machinery*, 13:547–569, 1966.

[6] B. Clarke. Discussion of the papers by Rissanen, and by Wallace and Dowe. *Computer Journal*, 42(4):338–339, 1999.

[7] J. W. Comley and D. L. Dowe. MDL and Minimum Message Length. In P. Grünwald, I. J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications.* MIT Press, 2004. Expected to be published mid-2004.

[8] A. P. Dawid. Discussion of the papers by Rissanen and by Wallace and Dowe. *Computer Journal*, 42(4):323–326, 1999.

[9] D. L. Dowe, R. A. Baxter, J. J. Oliver, and C. S. Wallace. Point Estimation using the Kullback-Leibler Loss Function and MML. In *Proc. 2nd Pacific Asian Conference on Knowledge Discovery and Data Mining (PAKDD'98)*, pages 87–95, Melbourne, Australia, April 1998. Springer Verlag.

[10] D. L. Dowe, J. Oliver, T. Dix, L. Allison, and C. Wallace. A decision graph explanation of protein secondary structure prediction. In *Proc. 26th Hawaii International Conference on System Sciences, Maui, Hawaii, Vol I*, pages 669–678, January 1993.

[11] D. L. Dowe and C. S. Wallace. Kolmogorov Complexity, Minimum Message Length and Inverse Learning. In *Proc. 14th Australian Statistical Conference (ASC-14)*, page 144, Gold Coast, Qld., Australia, July 1998.

[12] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society, Series B*, 14:107–114, 1952.

[13] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:4–7, 1965.

[14] L. Kornienko, D. Dowe, and D. Albrecht. Message length formulation of support vector machines for binary classification - a preliminary scheme. In *Proc. 15th Australian Joint Conference on Artificial Intelligence, Canberra, Australia, 2-6 Dec. 2002*, pages 119–130, 2002. Lecture Notes in Artificial Intelligence (LNAI) 2557, Springer Verlag, 2002.

[15] G. McLachlan and K. Basford. *Mixture Models.* Marcel Dekker, New York, 1988.

[16] J. Oliver. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. Extended version available as TR 173, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia.

[17] J. Oliver and C. Wallace. Inferring decision graphs. In *Proceedings of Workshop 8 — Evaluating and Changing Representation in Machine Learning IJCAI-91*, 1991.

[18] J. R. Quinlan. C5.0. http://www.rulequest.com.

[19] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the Minimum Description Length Principle. *Inform. Comput.*, 80(3):227–248, Mar. 1989.

[20] J. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[21] J. J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society (Series B)*, 49:260–269, 1987.

[22] J. J. Rissanen. Hypothesis selection and testing by the MDL principle. *Computer Journal*, 42:223–239, 1999.

[23] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach.* Prentice Hall, 1995.

[24] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Technical Jrnl.*, 27:379–423, 623–656, 1948.

[25] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22,224–254, 1964.

[26] R. J. Solomonoff. Two kinds of probabilistic induction. *Computer Journal*, 42(4):256–259, 1999.

[27] P. J. Tan and D. L. Dowe. MML inference of decision graphs with multi-way joins. In R. McKay and J. Slaney, editors, *Proc. 15th Australian Joint Conference on Artificial Intelligence - Lecture Notes in Artificial Intelligence*, number 2557, pages 131–142. Springer Verlag, December 2002.

[28] C. S. Wallace. False Oracles and SMML Estimators. In D. Dowe, K. Korb, and J. Oliver, editors, *Proceedings of the Information, Statistics and Induction in Science (ISIS) Conference*, pages 304–316, Melbourne, Australia, August 1996. World Scientific. Was Tech Rept 89/128, Dept. Comp. Sci., Monash Univ., Australia, June 1989.

[29] C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Journal*, 11:185–194, 1968.

[30] C. S. Wallace and D. M. Boulton. An invariant Bayes method for point estimation. *Classification Society Bulletin*, 3(3):11–34, 1975.

[31] C. S. Wallace and D. L. Dowe. Intrinsic classification by MML – the Snob program. In C. Zhang and et al., editors, *Proc. 7th Australian Joint Conf. on Artif. Intelligence*, pages 37–44. World Scientific, Singapore, 1994.

[32] C. S. Wallace and D. L. Dowe. Minimum Message Length and Kolmogorov Complexity. *Computer Journal*, 42(4):270–283, 1999. Special issue on Kolmogorov Complexity.

[33] C. S. Wallace and D. L. Dowe. MML clustering of multi-state, Poisson, von Mises circular and Gaussian distributions. *Statistics and Computing*, 10(1):73–83, January 2000.

[34] C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *J. Royal Statistical Society (Series B)*, 49:240–252, 1987.

[35] C. S. Wallace and K. B. Korb. Learning linear causal models by MML sampling. In A. Gammerman, editor, *Causal Models and Intelligent Data Management*, pages 89–111. Springer Verlag, 1999.

[36] C. S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.

[37] H. Wettig, P. Grünwald, T. Roos, P. Myllymäki, and H. Tirri. Supervised learning of Bayesian network parameters made easy. In *Proceedings of the Benelearn 2002 Conference, Utrecht, The Netherlands, Dec. 2002*, 2002.