LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-339

INFERRING DECISION TREES USING THE MINIMUM DESCRIPTION LENGTH PRINCIPLE

L. Ross Quinlan
Ronald L. Rivest

September 1987

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Inferring Decision Trees Using the Minimum Description Length Principle

by J. Ross Quinlan and Ronald L. Rivest

September 4, 1987

Abstract

We explore the use of Rissanen's Minimum Description Length Principle for the construction of decision trees. Empirical results comparing this approach to other methods are given.

1 Introduction

This paper concerns methods for inferring decision trees from examples for classification problems. The reader who is unfamiliar with this problem may wish to consult J.R. Quinlan's paper [8], or the excellent monograph by Breiman, Friedman, Olshen, and Stone [3], although this paper will be self-contained.

This work is inspired by Rissanen's work on the Minimum Description Length Principle (or MDLP for short) [10,11], and on his related notion of the stochastic complexity of a string [12]. The reader may also want to refer to related work by Boulton and Wallace [15,1,2], Georgeff and Wallace [4], and Hart [5].

Roughly speaking, the Minimum Description Length Principle states that the best "theory" to infer from a set of data is the one which minimizes the sum of

1. the length of the theory, and
2. the length of the data when encoded using the theory as a predictor for the data.

Here both lengths are measured in bits, and the details of the coding techniques are relevant. The encoding scheme used to encode the allowable theories and data reflect one's *a priori* probabilities.

This paper prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and a grant from the Siemens Corporation.

First author's addresses: School of Computing Sciences, New South Wales Institute of Technology, Sydney, N.S.W. 2007 Australia (*munnari!sesg.oz!quinlan@seismo.css.gov*)

Second author's addresses: MIT Lab. for Computer Science, Cambridge, Massachusetts 02139 USA (*rivest@theory.lcs.mit.edu*)

This paper explores the application of the MDLP to the construction of decision trees from data. This turns out to be a reasonably straightforward application of the MDLP. It is also an application area that was foreseen by Rissanen. ("...the design of of an optimal size decision tree can rather elegantly be solved by this approach without the usually needed fudge factors and arbitrary performance measures." [12, page 15])

The purpose of the present paper is thus to examine closely this proposal by Rissanen, to work out some of the necessary details, and to test the approach empirically against other methods. This paper may also serve as an expository introduction to the MDLP for those who are unfamiliar with it; but the interested reader is strongly encouraged to consult Rissanen's fascinating papers on these subjects [10,11,12] (and his papers referenced therein).

We formalize the problem of inferring a decision tree from a set of examples as follows. We assume that we are given a data set representing a collection of *objects*. The objects are described in terms of a collection of *attributes*. We assume that we are given, for each object and each attribute, the *value* of that attribute for the given object. In this paper we do not consider the possibility that some values may be *missing*; the reader should consult Quinlan [8] for advice on handling this situation.

We are also given, for each object, a description of the *class* of that object. The classification problem is often *binary*, where each object represents either a *positive* instance or a *negative* instance of some class. However, we will also consider non-binary classification problems, where the number of object classes is an arbitrary finite number. (As an example, consider the problem of classifying handwritten digits.)

Table 1 gives an example of a small data set, copied from [8]. Here the attributes are for various Saturday mornings, and the classification is positive if the morning is suitable for some "unspecified activity".

From the given data set, a *decision tree* can be constructed. A decision tree for the data in Table 1 is given in Figure 1. We can view the decision tree as a classification procedure. Some of the nodes (drawn as solid rectangles) are *decision nodes*; these nodes specify a test that one can apply to an object. The possible answers are the labels of the arcs leaving the decision node. In Figure 1, the tests simply name the attribute to be queried; the arcs give the possible values for the attribute. The dashed boxes of the figure are the *leaves* of the decision tree.

A decision tree defines a classification procedure in a natural manner. Any object (even one not in the original data set) is associated with a unique leaf of the decision tree. This association is defined by a procedure that begins at the root and traces a path to a leaf by following the arcs that correspond to the attributes of the object being classified. For example, object 10 of Table 1 would be associated with the rightmost leaf of the decision tree of Figure 1, since it has a rainy outlook but is not windy. A decision procedure with c leaves partitions the space of objects into c disjoint *categories*.

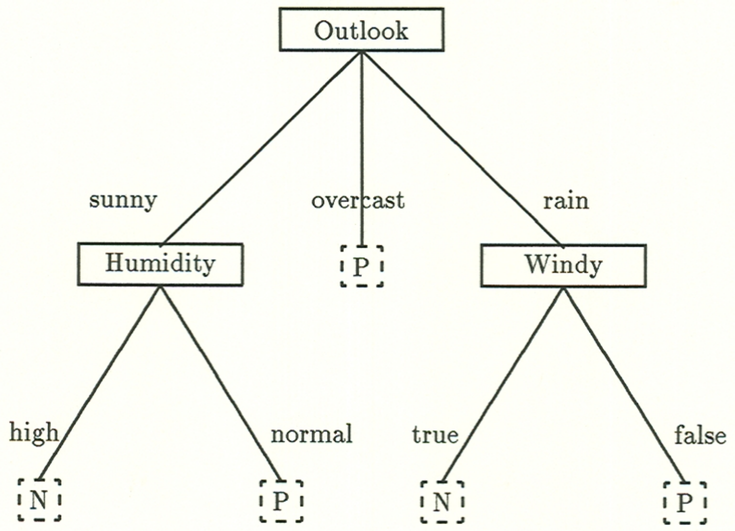

Figure 1: A Decision Tree

No.	Attribute				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

Table 1: A Small Data Set

With each leaf the decision tree associates a *class*; this the default class assigned by the decision tree to any object in the category associated with that leaf. New objects will be classified according to the default class of their category.

In our example, the available attributes are adequate to construct a decision tree which predicts the class perfectly (a *perfect* decision tree). In some cases, the objects in a given category may not all be of the same class. This may happen if the input data is noisy, if the given attributes are inadequate to make perfect predictions (i.e., the class of the given objects can not be expressed as a function of their attribute values), or if the decision tree is small relative to the complexity of the classification being made. If the decision tree is not perfect, the default class label for a leaf is usually chosen to be the most frequent class of the objects known to be in the associated category.

The problem is to construct the "best" decision tree, given the data. Of course, what is "best" depends on how one plans to use the tree. For example, a tree might be considered "best" if:

1. It is the smallest "perfect" tree.

2. It has the smallest possible error rate when classifying previously unseen objects.

In this paper we are primarily concerned with objective (2).

For this purpose, it is well known that it is not always best to construct a perfect decision tree, even if this is possible with the given data. One often achieves greater accuracy in the classification of new objects by using an imperfect, smaller decision tree rather than one which perfectly classifies all the known objects [3]. The reason is that a decision tree which is perfect for the known objects may be overly sensitive to statistical irregularities and idiosyncrasies of the given data set.

However, it is generally not possible for a decision tree inference procedure to *explicitly* minimize the error rate on new examples, since the "real world" probability distribution generating the examples may be unknown or may not even exist.

Consequently, a number of different approximate measures have been proposed for this purpose. For example, Quinlan [8] studies some information-theoretic measures similar to the MDLP in spirit. The MDLP is an approximate criteria in the same sense: minimizing the appropriate "description-length" (to be defined) can be viewed as an attempt to minimize the "true" error rate for the classification procedure.

Another level of approximation usually arises because it is usually infeasible in practice to determine which decision tree actually minimizes the desired measure. There are just too many candidate decision trees, and there seems to be no efficient way of identifying the one which optimizes the chosen measure. Thus one is forced to adopt heuristics here as well which attempt to find a tree that is good or near-optimal with respect to the chosen measure. A commonly used heuristic is to build a large tree in a top-down manner, and then to iteratively prune leaves off until a tree is found that seems to minimize the desired measure [3]. It is not uncommon for *different* criteria to be used during the pruning phase than during the initial building phase. Growing an overly-large initial tree will often allow dependencies between the attributes to be discovered which might not reveal themselves quickly enough if an attempt was made to grow the tree top-down until it seemed that the measure was minimized.

No matter what search technique is used to find a good tree according to the desired measure, the choice of the approximate measure itself can have a large effect on the quality of the resulting decision tree.

2 The Minimum Description Length Principle

In this section, we describe how Rissanen's Minimum-Length Description Principle [10,11,12] naturally defines a measure on decision trees (relative to a given set of data), where the decision tree which minimizes this measure is proposed as a "best" decision tree to infer from the given data.

We will motivate the Minimum-Length Description Principle by considering a communication problem based on the given data. The Minimum Description Length Principle will define the "best" decision tree to be the one that allows us to solve our communication problem by transmitting the fewest total bits. Of course, this communication problem is just an artifice used in the *definition* of the "best" tree; the *real* objective is to produce that decision tree which will have the greatest accuracy when classifying new objects.

Our communication problem is the following. You and I have copies of the data set (e.g. Table 1), but in your copy the last column – giving the class of each object – is missing. I wish to send you an exact description of the missing column using as few bits as possible. We agree in advance on an encoding technique to be used to transmit the missing column to you with. The simplest technique would be for me to transmit the column itself to you directly. In our example this would require exactly 14 bits, independent of what classifications the objects have.

However, if the class of an object depends to any significant extent on its attributes, then I may be able to dramatically reduce the number of bits I need to send, if we have agreed to use an encoding technique that allows me to express such dependencies. For example, suppose it sufficed for me to say, "an object is in the positive class if and only if it has high humidity". This would require only a few bits, independent of the size of the table.

In general, the more predictable that the class of an object is from the object's attributes, the fewer bits I may need to send in order to communicate to you the missing class column. To this end, it may be helpful for us to agree on an encoding technique that allows reference to various subsets of the objects defined by their attributes, such as "all windy high humidity objects". Since both of us know the attributes of each object, you can determine which objects I am referring to when I use such descriptions.

In general, I may find it worthwhile to:

1. Partition the set of objects into a number of subsets or categories, based on the attributes of the objects.

2. Send you a description of this partition.

3. Send you a description of the most frequent (or default) class to be associated with each subset.

4. For each category of objects, send you a description of the *exceptions* – by naming those objects in the category whose actual classification is different than the default class, together with the correct classification for those objects.

This may be worthwhile since if there are few exceptions in a category, only a few bits will be needed to describe them. Although I need to use some bits in order to describe to you the partition, this partition may more than pay for its cost by means of the data compression I can later achieve in step 4.

A natural and efficient way of partitioning the set of objects into disjoint categories, and associating a default class with each category, is to use a decision tree. This is the approach we will use in this paper.

The "best" decision tree, for our communication problem, is *defined* to be the one which enables me to send you the fewest possible bits in order to describe to you the missing class column in your table. For this tree, the *combined length* of the description of the decision tree, plus the description of the exceptions, must be as small as possible. Of course, the actual cost will depend on the methods used to encode the decision tree and the exceptions – more about this later.

This "optimal" (according to the MDLP) decision tree can then be used to classify new objects.

The communication problem defined above captures the essence of Rissanen's Minimum Description Length Principle. The "best" tree for the communication problem is proposed as the "best" tree to infer from the given data. Dependencies between an object's class and its attributes which are pronounced and prevalent enough to allow me to save bits in the communications problem are judged to be significant and worth including in the inferred decision tree. Dependencies which are weak or are only represented in a few cases are judged to be insignificant, and are omitted from the tree. The communication problem thus provides a mathematically clean and rigorous way of defining the "best" decision tree to infer from a given set of data, relative to the method used to encode the tree and the exceptions. Furthermore, since coding length and predictability are intimately related, one has reason to expect that such a decision tree will do well at classifying new, unseen cases (see [11,12]).

2.1 A Bayesian Interpretation of the MDLP

In the next section we turn to the question of coding techniques. Before doing so, we point out that the MDLP can be naturally viewed as a Bayesian MAP (Maximum A Posteriori) estimator. Let T denote our decision tree, and let t denote its length in bits when encoded as described in the next section. Similarly, let D denote the data to be transmitted (the last column of the object description table), and let d denote its length when encoded as described in the next section, using the tree to describe all the "non-exceptional" classes.

Let r be a fixed parameter, $r > 1$. (In typical usage, $r = 2$.) We associate with each binary string of length t (here $t \geq 0$) the probability

$$(1 - \frac{1}{r})(\frac{1}{2r})^t \tag{1}$$

so λ (the empty string) has probability $(1 - \frac{1}{r})$, the strings 0 and 1 each have probability $(1 - \frac{1}{r})(\frac{1}{2r})$, and so on. It is easy to check that the total probability assigned to strings of length t is $(1 - \frac{1}{r})(\frac{1}{r})^t$ and that the total probability assigned to all strings is 1. The parameter r controls how quickly these probabilities decrease as the length t of the string increases; as r increases these probabilities decrease more quickly. This procedure allows us naturally to associate a probability with a string.

Let r_T and r_D be two fixed parameters with $r_T > 1$ and $r_D > 1$. Then we can interpret the Minimum Description Length Principle in a Bayesian manner as follows, using the above procedure for associating probabilities with strings.

1. The length t of encoding of the tree T is used to determine the *a priori* probability of the theory represented by the decision tree:

$$P(T) = (1 - r_T)(\frac{1}{2r_T})^t \tag{2}$$

2. The length d of the data is used to determine the conditional probability of the observed data, given the theory:

$$P(D \mid T) = (1 - r_D)(\frac{1}{2r_D})^d \tag{3}$$

3. The negative of the logarithm of the *a posteriori* probability of the theory is by Bayes' Formula a linear function of t and d:

$$P(T \mid D) = \frac{P(D \mid T)P(T)}{P(D)} \tag{4}$$

   implies that

$$\begin{aligned} -\lg(P(T \mid D)) &= t(1+\lg(r_T)) + d(1+\lg(r_D)) \\ &\quad - \lg(1-r_T) - \lg(1-r_D) + f(D) \\ &= tc_T + dc_D + g(r_T, r_D, D) \end{aligned} \tag{5}$$

where $f(D)$ is a constant that depends on the data D but not on the tree T, and where $g(r_T, r_D, D)$ is a constant depending only on D and the parameters r_T and r_D, these constant values can thus be safely ignored when trying to find the best tree T for the data D. The tree which minimizes $tc_T + dc_D$ will have maximum *a posteriori* probability.

If, for example, we choose $r_T = r_D = 2$, then $c_T = c_D = 2$, and finding the best theory is equivalent to minimizing the sum $t + d$. Choosing other values for r_T and/or r_D will give rise to other linear combinations of t and d. If r_T is large, then large trees T will be penalized more heavily, and a more compact tree will have maximum *a posteriori* probability. In the limit, as $r_T \to \infty$, the resulting tree will be the trivial decision tree consisting of a single node giving the most common class among the given objects. If r_D is large, then a large tree, which explains the given data most accurately, is likely to result, since exceptions will be penalized heavily. In the limit, as $r_D \to \infty$, the resulting decision tree will be a perfect decision tree, if one exists. Thus, choosing r_T and r_D amount to choosing one's *a priori* bias against large trees or large numbers of exceptions.

In the rest of this paper, unless stated otherwise, we will assume that $r_T = r_D$, so that $c_T = c_D$, and we will wish to minimize $t + d$; this corresponds to the Minimum Description Length Principle in its simplest form.

One can view the contribution of the Minimum Description Length Principle, in comparison with a Bayesian approach, as providing the user with the conceptually simpler problem of computing code lengths, rather than estimating probabilities. It is easier to think about the problem of coding a decision tree than it is to think about assigning an *a priori* probability to the tree.

3 Choice of coding techniques

There are many different techniques one could use to encode the decision tree and the exceptions. In this section we propose some particular techniques for consideration. For a given set of data, and for each possible encoding technique, a "best" tree can be computed (in principle, although in practice it may be difficult to compute such a "best" tree).

It is important that the encoding techniques chosen be efficient. An inefficient method of encoding trees will cause decision trees which are too small to be produced, since the "tree" portion of our communication cost will be too high. Symmetrically, an inefficient method for encoding exceptions will tend to result in overly large trees being produced.

This paper suggests some particular encoding techniques. The utility of the Minimum Description Length Principle is not based on the use of any particular techniques.

The Minimum Description Length Principle provides a way of comparing decision trees, once the encoding techniques are chosen.

4 Details of Coding Methods

In order to illustrate the details of the approach suggested above, we outline techniques for coding messages, strings, and trees in this section.

In this paper, all logarithms are to the base 2; we denote the base two logarithm of n as $\lg(n)$.

4.1 Coding A Message Selected from a Finite Set

We shall need ways to encode a message that is selected from a finite set.

If the message to be transmitted is selected from a set of n *equally likely* messages, then $\lg(n)$ bits are required to encode the selected message. (In this paper we shall generally ignore the issues that arise concerning the use of non-integral numbers of bits. The use of techniques such as arithmetic coding [14] can justify using non-integral numbers, rather than rounding up; arithmetic codes can be as efficient as the non-integral numbers indicate, when many messages are being sent. Also, we are less interested here in actually coding the data than in knowing how much information is present.)

If the messages have unequal likelihoods which are known to the receiver, then $-\lg(p)$ bits are required to transmit a message which has probability p, using an ideal coding scheme. Of course, if the n messages are equally likely, this reduces to our previous measure.

4.2 Coding Strings of 0's and 1's

We shall also need techniques for encoding finite-length strings of 0's and 1's.

In particular, we are interested in the problem of transmitting a string of 0's and 1's so that it will be cheaper to transmit strings which have only a few 1's. (The ones will indicate the location of the exceptions.) We assume that the string is of length n, that k of the symbols are 1's and that $(n-k)$ of the symbols are 0's, and that $k \leq b$, where b is a known *a priori* upper bound on k. Typically we will either have $b = n$ or $b = (n+1)/2$.

The procedure we propose is:

- First I transmit to you the value of k. This requires $\lg(b+1)$ bits. (See Appendix A for a variation on this proposal.)
- Now that you know k, we both know that there are only $\binom{n}{k}$ strings possible. Since all these possible strings are equally likely *a priori*, I need only $\lg(\binom{n}{k})$ additional bits to indicate which string actually occurred.

The total cost for this procedure is thus

$$L(n,k,b) = \lg(b+1) + \lg(\binom{n}{k}) \text{ bits} \qquad (6)$$

When we are transmitting the location of exceptions for a binary classification problem, we will have $b = (n+1)/2$; in several other cases we will have $b = n$.

We may consider coding in this manner the string in the last column of table 1:

$$N, N, P, P, P, N, P, N, P, P, P, P, P, N \qquad (7)$$

Treating N as 0 and P as 1, we have $n = 14$, $k = 9$, and $b = 15$, for a total of

$$L(14, 9, 14) = \lg(15) + \lg(2002) = 14.874 \text{ bits.}$$

This is larger than the "obvious" cost of 14 bits; this coding scheme can save substantially when k is small, in return for an increased cost in other situations (as in the present example).

We propose using $L(n,k,b)$ as the standard measure of the complexity of a binary string of length n containing exactly k 1's, where $k \leq b$. This is an accurate measure of the number of bits needed to transmit such a string using the proposed scheme.

The formula for $L(n,k,n)$ is also derivable by another coding method, which we sketch here. (This method and analysis are due to Rissanen [11].) I will transmit 0's and 1's to you one by one. However, after I have transmitted t symbols to you, s of which are 1's, we shall consider the probability of the next symbol as being a 1 as $(s+1)/(t+2)$ – this is Laplace's famous "Rule of Succession". Similarly, the probability of the next symbol being an 0 is considered to be $((t-s)+1)/(t+2)$. This can be viewed as a straight frequency ratio, where the initial values for the number of 0's and the number of 1's seen so far begin at *one* each rather than *zero*. For example, the initial estimated likelihood of seeing a 1 is $1/2$, and the likelihood of seeing an 0 as the second symbol if the first symbol was a 1 is $1/3$. At each step, the probabilities of 0's and 1's are computable, and these probabilities are used in the coding, so that a symbol of probability p only requires $-\lg(p)$ bits to represent. With a little algebra, one can prove that the number of bits needed to represent a string of n symbols containing k 1's using this technique is exactly $L(n,k,n)$.

The function $L(n,k,b)$ can be approximated using Stirling's formula to obtain:

$$L(n,k,b) = nH(k/n) + \frac{4\lg(n)}{2} - \frac{\lg(k)}{2} - \frac{\lg(n-k)}{2} - \frac{\lg(2\pi)}{2} - \lg(b) + O(1/n). \quad (8)$$

where $H(p)$ is the usual "entropy function":

$$H(p) = -p\lg(p) - (1-p)\lg(1-p). \quad (9)$$

It is interesting to note that $L(n,k,b)$ does not depend on the position of the k 1's within the string of length n; any string of length n which contains exactly k 1's will be assigned a codeword of length exactly $L(n,k,b)$ bits. In our application, where the order of the objects in the table is arbitrary, this seems appropriate.

Quinlan's heuristic [8] is based on related ideas; he measures the information content in a string of length n containing k P's as $nH(k/n)$. The use of this underapproximation to $L(n,k,b)$ may result overly large decision trees, by our standards. In addition, he does not consider the cost of coding the decision tree at all; his method may be viewed as a maximum-likelihood technique rather than a MAP technique.

We note that the natural generalization of this method to nonbinary classification problems would assign a cost of

$$L(n;k_1,k_2,\ldots,k_t) = \lg\left(\binom{n+k-1}{k-1}\cdot\binom{n}{k_1,k_2,\ldots,k_t}\right) \quad (10)$$

to a string of length n containing k_1 objects of class 1, ..., k_t objects of class t, where $k = k_1 + \ldots + k_t$. Here the upper bound b on the k_i's is omitted and assumed to be n.

There are of course a number of different variations one could try. Each such variation corresponds to a different "model class" or choice of prior probabilities for our representation of strings. Appendix A describes one technique which encodes small values of k more compactly than our standard scheme. An even more highly biased scheme would encode 0 as 0 and $k > 0$ as 1^k0.

4.3 Coding Sets of Strings

In our example, I might partition the objects into those with "high humidity", and those with "normal humidity". This results in the final column being divided into two parts:

$$N, N, P, P, N, P, N \text{ for the high humidity objects} \tag{11}$$

where the default class is "N", and

$$P, N, P, P, P, P, P \text{ for the normal humidity objects.} \tag{12}$$

where the default class is "P". To code the exceptions will require only

$$L(7,3,3) + L(7,1,3) = 11.937 \tag{13}$$

bits. Since this is less than the "obvious" coding length of 14 bits, there seems to be some relationship between the attribute "humidity" and the class of the object. The complexity of representing the exceptions has been reduced by breaking it into two parts.

Of course, we would also need to include the the cost of describing this simple decision tree (containing only one decision node), before we can decide if such a partition is worthwhile.

4.4 Coding Decision Trees

How can I code a decision tree efficiently?

It seems natural to use a coding scheme where smaller decision trees are represented by shorter codewords than larger decision trees.

We assume for now that the attributes have only a finite number of values, as in our example. We discuss countable or continuous-valued attributes later.

Our procedure for encoding the decision tree is a recursive, top-down, depth-first procedure.

A leaf is encoded as a "0" followed by an encoding of the default class for that leaf.

To code a tree which is not a leaf, we begin with a "1", followed by the code for the attribute at the root of the tree, followed by the encodings of the subtrees of the tree, in order. If the root attribute can have v values, then the code for the tree is obtained by concatenating the codes for the v subtrees after the code for the root.

This procedure is applied recursively to encode the entire tree.

If there are four possible attributes at the root, we need two bits to code the selected attribute. However, note that attributes deeper in the tree will be cheaper to code, since there are fewer possibilities remaining to be used deeper in the tree.

As an example, the code for the tree of Figure 1 would be:

```
1 Outlook 1 Humidity 0 N 0 P 0 P 1 Windy 0 N 0 P
```

This corresponds to a depth-first traversal of the tree, where 0's indicate leaves (with following default class) and a 1's indicate decision nodes (with following attribute name). The substring "`1 Humidity 0 N 0 P`" corresponds to the left subtree of the root, the substring "`0 P`" corresponds to the middle subtree, and the substring "`1 Windy 0 N 0 P`" corresponds to the right subtree. Here the code for "Outlook" would indicate that we are selecting the first attribute out of four, so this would require two bits. On the other hand, the code for "Humidity" would require only lg(3) bits, since there are only three attributes remaining at this point in the tree, since Outlook is already used. The example tree requires 18.170 bits to encode.

The proposed encoding technique above for representing trees is nearly optimal for binary trees, but is not so good for trees of higher arity. In general, a uniform b-ary tree with n decision nodes and $(b-1)n+1$ leaves will require $bn+1$ bits using our scheme, (not counting the bits required to encode the attribute names or default classes), whereas the number of b-ary trees with n internal nodes and $(b-1)n+1$ leaves is (see [7, Exercise 2.3.4.4.11])

$$\frac{1}{(b-1)n+1}\binom{bn}{n}; \tag{14}$$

the base two logarithm of which is

$$bnH(\frac{1}{b}) + \frac{\lg(bn)}{2} - \frac{\lg((b-1)n)}{2} - \frac{\lg(n)}{2} - \frac{\lg 2\pi}{2} + o(1). \tag{15}$$

where $H(p)$ is the usual entropy function (using base two logarithms). Even counting the extra bits required to specify the size of the tree, the proposed coding scheme is not as efficient for high arity trees as one might desire.

To fix this, the following approach can be used. Consider the bit string representing the structure of the tree (i.e. excluding the attribute names and default classes). For binary trees this string contains nearly as many ones as zeros, whereas for trees using attributes of high arity there will be many more zeros than ones. Suppose the tree has k decision nodes and $n-k$ leaves. Then the tree's description string will be of length n and will contains k ones. Note that $k < n-k$ since all tests will have arity at least two. Thus *we should specify the cost of describing the structure of the tree as* $L(n, k, (n+1)/2)$. To obtain the total tree description cost, we then add in the cost of specifying the attribute names at each node, and the cost of specifying the default class for each leaf, using the cost measures previously described.

There are several ways one can improve upon the above coding technique. A simple example is to note that in some cases the default class of a leaf is obvious. (If the classification problem is binary, the leaf is the right child, and the other child is a leaf, then the default class for the leaf must be the complement of its sibling's default class, otherwise the decision is useless.) We do not pursue these approaches here.

4.5 Coding Exceptions

In addition to coding the decision tree, I need to code the exceptional objects whose classes are different than the default classes of their categories.

For binary classification problems, this is relatively straightforward, since all I need to do is to indicate the *positions* of the exceptions.

We prefer to do so on a category-by-category basis, since this works most smoothly with our procedures for growing a good decision tree. There are other obvious candidate encoding schemes – such as coding up the locations of the exceptions in a global manner – which may be more efficient as coding techniques overall but which are more difficult to integrate into search procedures for good trees.

Let us return to our example. Given our example decision tree, we have divided the set of objects into five subsets:

```
sunny outlook & high humidity:    N, N, N
sunny outlook & normal humidity:  P, P
overcast outlook:                 P, P, P, P
rainy outlook & windy:            N, N
rainy outlook & not windy:        P, P, P
```

The exceptions (there are none) can be encoded with a cost:

$$L(3,0,1) + L(2,0,1) + L(4,0,2) + L(2,0,1) + L(3,0,1) = 5.585 \text{ bits} \qquad (16)$$

The total cost for our communication problem using the example tree is thus 18.170 bits for the tree, plus 5.585 bits for the exceptions — a total cost of 23.754 bits.

For non-binary classification problems, we propose coding the exceptions using an iterative approach within each category (assuming the default class for the category has already been coded in the structure of the tree):

- Identify the locations of the exceptions.
- Identify the most common class occurring among the exceptions; this is the "first alternative class" for that category.
- Identify the locations of the "second-order" exceptions within the exceptions; these objects are neither default class for the category nor the first alternative class for that category.

- Iterate as necessary with higher-order exceptions and higher-order alternative classes until no further exceptions remain.

4.6 Coding Real-Valued Attributes

For real-valued attributes (such as age or weight) we must modify our coding techniques. The approach we propose is to find a good "cut point"; a decision node will not only name the attribute (e.g. age) but also the value of the cut point (e.g. 40), so that the decision will be a *binary* decision of the form "Is $age < 40$?". In computing the length of the description of the decision tree, we will need to explicitly measure the cost of representing the value of the cut point.

There are two approaches that come to mind:

- *Using values of the known objects:* Suppose that for the desired attributed the n given objects have $m \leq n$ distinct values. A decision node can specify a real-valued cut-point by sorting the m real values associated with the known objects, and specifying the i-th such number by specifying i. Although one could merely specify i using $\lg(m)$ bits, it seems preferable to use some short encodings to represent a well-distributed set of i's. One such approach is to order the fractions i/m so that we first have an approximation to 1/2, then approximations to 1/4 and 3/4, then good approximations to 1/8, 3/8, 5/8, 7/8, and so on. The j-th such approximation is represented by coding j using only $\mathrm{lb}(j)$ bits (see Appendix A); this represents the i-th largest value of the attribute on the given data. A second approach is to select approximately $\sqrt{m}$ evenly-spaced values from the sorted list of values, and to use $\lg(\sqrt{m})$ bits to indicate which one to use as a cut-point. For a justification of a very similar approach, see Wallace and Boulton's paper [15].

- *Using compactly described rational numbers:* A binary rational fraction with numerator a and denominator 2^b can be represented by the pair of integers (a, b). The coding techniques described, for example, in Appendix A, can be used to encode these integers. It may not pay to use a high-precision number in a test if a simpler number performs nearly as well.

Although we have experimented with these approaches, it is difficult to distinguish their performance; for definiteness, let us propose the second method.

5 Discussion

We now have a well-defined procedure for me to use to communicate the class column to you. I will pick the decision tree which "pays for itself" in terms of the data compression it permits by coding the induced subset separately.

The decision tree I pick will be a good decision tree for the data (relative to the coding method selected). It reflects important structure in the relationship between the attributes and the class of the objects, but won't contain decisions whose effect isn't strong enough to justify their inclusion in the tree. The communication cost measure provides a rationale for picking the right intermediate amount of structure.

6 Computing Good Decision Trees

It is probably difficult to compute the *best* decision tree under our measure. Hyafil and Rivest [6] prove that constructing an optimal binary decision tree is NP-complete when the cost of a tree is its external path length; it may be possible to modify this proof to handle the current situation, although we haven't done so.

Heuristics for growing good trees in a top-down manner derive naturally from the discussion of the previous section.

The incremental cost of replacing a leaf with a decision node is easily measured. Suppose there are A attributes altogether in our problem, and we are considering replacing a leaf at depth d with a decision node based on some attribute. Suppose that on the path from the root to this leaf at depth d, there are $d' \leq d$ discrete attributes tested. These attributes can not be tested again, so that d' of the A attributes are not eligible for use in this position. Thus, there are $A - d'$ attributes eligible, and to indicate which one is selected will require $\lg(A - d')$ bits. If the attribute selected has v possible values, the rest of the incremental cost for describing the additional tree structure (exclusive of the attribute name but including the new default classes) is $2v - 1$ bits.

This operation splits one subset into v subsets. We can measure the extent to which the exceptions can now be coded more efficiently. If the savings so obtained is greater than the cost of extending the tree, the extension should be selected. If several such extensions are possible, we can pick the one that yields the greatest net savings.

We propose a two-phase process for growing a good decision tree.

In the first phase we begin with the null tree (a single leaf), and continue to extend the tree by iterating the following procedure until the tree is perfect or can't be grown any further:

1. Let x be a leaf whose corresponding category of objects are of varying classes, such that it is possible to replace x with a decision node (i.e. x is not at maximum possible depth).

2. For each possible attribute A that might be specified in the decision node to replace x, compute the total communication cost if this change is made. (The cost is the resulting description length for the tree plus exceptions.

Note that only a portion of the code is changed, making this computation a local one.)

3. Replace x with the decision node having least total communication cost. (Note that the total communication cost may go up.)

During the second phase, the tree is repeatedly pruned back by replacing decision nodes (all of whose children are leaves) by leaves, whenever this improves the total communication cost, until no further improvement in communication cost is possible.

One can easily imagine more elaborate search procedures which work harder to find better decision trees under our complexity measure. For example, we could select the attribute (if any) to use in a given decision node by explicitly considering the quality of all of the depth zero, one, or two decision subtree trees that can be built at that position. The attribute at the root of the best such subtree would be selected as the attribute to use at that position, and the process would begin over to select the attributes (if any) to use at the positions in the next level of the tree.

We note that choosing an attribute with a large number of values is penalized using the cost measure suggested here. This addresses one of the difficulties raised by Quinlan. With many previous methods of growing a decision tree, an attribute with many values would have a high chance of being chosen, since it split the set of objects into many subsets. One can imagine, for example, using the "object number" attribute (the first column of our Table 1) at the root of the decision tree(!). Although there is no real "structure" here, using this attribute in our example allows the object set to be divided into sets which are purely of one class or of the other (i.e. the individual objects). This approach is penalized using the MDLP, since the cost of specifying the attribute won't be justified in terms of the extra compression achieved in transmitting the class information.

It is interesting to note that for the small example given above, the "humidity" attribute is the most promising according to our measure, whereas Quinlan selected the "Outlook" measure for the root of the tree. However, with our measure we find that even the "Humidity" decision node is too costly; it is better to have the trivial tree consisting of just one leaf. The given example is really just a toy example because of its small size, and our approach realizes this by noting that for this example, no nontrivial tree is worth paying for. For this example, our use of the MDLP would suggest that given the available data, it is best to guess that the class of an unseen object is "P", independent of its attributes. (Recall that our stated objective is to be able to classify new, unseen, objects well, *not* to classify the given objects perfectly.)

	MDLP		C4	
Data Set	Size	Error Rate	Size	Error Rate
Hypo	11	0.6%	11.0	0.55%
Discordant	15	1.9%	13.6	1.25%
LED	83	26.9%	56.0	28.1%
Credit	14	17.4%	32.5	16.1%
Endgame	15	17.9%	62.6	13.6%
Prob-Disj	17	20.5%	42.6	14.9%

Table 2: Experimental Results

7 Empirical Results

In order to test this approach of using the MDLP to guide the inference of decision trees from data, we implemented it and compared it with one of the best-performing alternative approaches. This approach is called "C4 with pessimistic pruning" and is described by Quinlan in [9]. We tested our approach on six data sets also used in [9]. We note that the C4 procedure is nondeterministic in nature, and we obtained our results by averaging over a number of trees grown with the same data set – hence the non-integral average tree sizes in the table. In contrast, our MDLP approach is deterministic, and only one tree is produced for a given set of data.

The "Size" columns represent the size of the average size of the decision tree computed (number of leaves). The "Errors" columns represent the percentage error rate of the decision trees constructed on new examples. These error rates were computed by constructing a decision tree using two-thirds of the available data as a training set, and then using the remaining one-third as a test set.

For those data sets which had real-valued attributes, the decision trees created used one of the known n values of that attribute as a cut-point, and charged the tree $\lg(n)$ bits for representing that cut point.

The trees created using the MDLP were grown out to the maximum possible size first, using MDLP to guide the selection of the attributes, and then pruned back in a way that minimized the description length.

We see that the trees created with the MDLP compare favorably with those created using the C4 technique with pessimistic pruning. In the "Hypo" data set, they created exactly the same decision tree. In general, the MDL method created smaller decision trees (the LED data set was the exception). However, for the last two problems the decision trees created by using the MDL are probably *too* small, resulting in a larger error rate. It is not unlikely that our coding techniques have some residual inefficiencies that are particularly relevant for these examples.

Overall, we are very encouraged by these experimental results. The MDLP provides a unified framework for both growing and pruning the decision trees, and these trees seem to compare favorably with those created by other high-performance techniques. Moreover, we believe that there certainly is room for further study and refinement of our methods, which will very likely result in even better overall performance.

8 Extensions

The proposed procedure for growing decision trees should work well in the presence of noisy data. As the noise level increases, the tree grown should decrease in size, since the significance of the existing dependencies of the class on the attributes will be masked.

The proposed procedure can easily be adapted to handle "training sets" which are especially representative of the concept being learned. One simple way to do so is to associate a "frequency count" larger than one with each input object in the data set. If it is desired that the decision tree produced will classify each object in the training set correctly, these counts can be set to a large value. (As the counts increase, the savings that can be realized by using a perfect decision tree increases relative to other decision trees.)

As a special case of the above notion, we can imagine replicating the data set some number c times. This is approximately equivalent to saying that if we separate my communication into t "tree bits" and d "data bits", that the total cost should be $t + cd$ rather than just $t + d$. As c increases, we can afford to record in our "best" tree ever more subtle dependencies. In the limit as $c \rightarrow \infty$, we can afford a perfect decision tree, if one exists. While the Minimum Description Length Principle suggests using $c = 1$, we can adjust c to reflect our a priori understanding of the representativeness or completeness of the given data set. In the notation of section 2.1 $c = c_D/c_T$; larger values of c say that as lengths increase the probabilities associated with the data strings decrease faster than the probabilities associated with the tree strings. Table 3 gives some experimental results for various values of c. At the moment we have little theory or justification for using values of c other than 1; however our empirical results suggest that using values of c somewhat greater than 1 may sometimes be advantageous. We leave this issue as an open problem.

9 Conclusion

We have discussed the application of Rissanen's Minimum Description Length Principle to the induction of decision trees. While the approach proposed here is close in spirit to that of Quinlan in looking at the information content of strings, we also charge for the cost of representing the tree itself. Our experimental results demonstrate the viability of this approach.

	c=1		c=2		c=8	
Data Set	Size	Error Rate	Size	Error Rate	Size	Error Rate
Hypo	11	0.6%	15	0.6%	19	0.5%
Discordant	15	1.9%	23	1.8%	37	1.1%
LED	83	26.9%	93	27.0%	95	27.0%
Credit	14	17.4%	11	13.5%	76	17.0%
Endgame	15	17.9%	35	11.5%	71	12.0%
Prob-Disj	17	20.5%	45	13.5%	69	13.0%

Table 3: Experimental Results for various r

10 Acknowledgements

We would like to thank Ed Puckett for introducing us to Rissanen's work, and for many helpful discussions regarding the Minimum Description Length Principle. We would also like to thank George Hart and Rick Lathrop for their thoughtful critiques of early drafts of this paper.

References

[1] D. M. Boulton and C. S. Wallace. An information measure for hierarchic classification. *The Computer Journal*, 16(3):254–261, August 1973.

[2] D. M. Boulton and C. S. Wallace. An information measure for single-link classification. *The Computer Journal*, 18(3):236–238, August 1973.

[3] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[4] M. P. Georgeff and C. S. Wallace. A general selection criterion for inductive inference. In *ECAI 84: Advances in Artificial Intelligence*, pages 473–482, Elsevier Science Publishers, 1984.

[5] George W. Hart. *Minimum Information Estimation of Structure.* PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, April 1987. Appears as LIDS-TH-1664.

[6] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

[7] Donald E. Knuth. *The Art of Computer Programming: Fundamental Algorithms.* Volume 1, Addison-Wesley, 1968.

[8] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[9] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 1987. (To appear.).

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

[11] Jorma Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.

[12] Jorma Rissanen. *Stochastic Complexity and Sufficient Statistics.* Technical Report, IBM Research Laboratory (San Jose), 1986.

[13] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[14] Jorma Rissanen and Glen G. Langdon, Jr. Universal modeling and coding. *IEEE Transactions on Information Theory*, IT-27(1):12–23, January 1981.

[15] C.S. Wallace and D.M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

Appendix A. A Variation on Coding Strings

In this section we describe a variation on the above procedure for coding strings which favors more strongly strings with high bias, and derive a different measure $L'(n,k,b)$ for the complexity of a string of length n containing exactly k 1's, where $k < b$. The idea is to code the integer k in such a way that smaller k's have shorter representations. This corresponds to a change in the probabilities assigned to data strings by the theories associated with the decision trees.

Consider coding a message k drawn from the set $\{0,1,\ldots,b\}$. We will also be interested in coding nonnegative natural numbers in a similar manner. The scheme proposed here is very close to the approach suggested by Rissanen [13] for the latter case, and "truncates" this approach to handle the former case. (Other schemes are of course possible.)

We define the cost $\mathrm{lb}_b(k)$ of coding an integer k chosen from the set $\{0,1,\ldots,b\}$ by the equations

$$\begin{aligned} \mathrm{lb}_b(0) &= 1 \\ \mathrm{lb}_b(k) &= 1 + \lg(k) + \lg(\lg(k)) + \lg(\lg(\lg(k))) + \ldots + C_b \end{aligned} \tag{17}$$

where the sum only includes those terms which are positive and where the constant C_b is chosen such that

$$\sum_{k=0}^{b} 2^{-\mathrm{lb}_b(k)} = 1. \tag{18}$$

b	C_b
1	0.000000
2	0.584963
3	0.774258
4	0.876024
9	1.024858
99	1.161168
999	1.198712
9999	1.218215
99999	1.230730
999999	1.239310
∞	2.865064

Table 4: Values for the constant C_b

(As a mnemonic, think of $\text{lb}_b(k)$ as the (l)ength of k in (b)its.) We also denote by $\text{lb}(k)$ the limit of $\text{lb}_b(k)$ as $b \to \infty$.

Note that in the above scheme, the number $k = 0$ requires only one bit to be coded.

We note a few values of C_b in Table 2 (the latter values of b are of the form $10^t - 1$ since a given value of b corresponds to selecting from a set of size $b+1$). Observe that the constant C_b approaches the limit $2.865064\ldots$ from below as $b \to \infty$ (the convergence is *very* slow; see Rissanen [13] for a derivation).

Given such a representation for integers k, we can represent a string of length n containing k 1's (where $k \leq b$) using only

$$L'(n,k,b) = \text{lb}_b(k) + \lg(\binom{n}{k}) \text{ bits.} \tag{19}$$

This coding scheme favors strings with high bias noticeably, since the coding of k is quite short for small k. In particular, if $k = 0$ or $k = n$, we will need only two bits.