RJ 8673 (78235) March 16, 1992
Computer Science

IBM Research Report

MDL-Based Pruning of 2^n-Tree Classifiers

Byron E. Dom
David Steele

K54/B2
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
e-mail: dom@almaden.ibm.com

Research Division
Almaden · T.J. Watson · Tokyo · Zurich

MDL-Based Pruning of 2^n-Tree Classifiers

Byron E. Dom and David Steele
IBM Almaden Research Center
650 Harry Rd.
San Jose, CA 95120-6099

ABSTRACT: This paper describes a technique for both decreasing the memory requirement and improving the performance of 2^n-tree classifiers (described in [1, 2]). This technique is an extension of that proposed by Quinlan and Rivest[5] for growing and pruning certain types of decision tree classifiers. The technique is based on the *Minimum Description Length principle* (MDL), proposed by Rissanen[7] as a measure of the goodness of models for describing data. In this technique the classification tree produced by the 2^n-tree training algorithms is considered to be a model for the class assignments of the training data conditioned on the associated feature vectors. A scheme is proposed for encoding the data using this model. This scheme consists of a code for the model (the 2^n-tree in this case) and a code for the class assignments conditioned on the model and the feature vectors. The latter consists of an encoding of the correct class codes for those training vectors misclassified by the 2^n classification tree. Associated with this encoding scheme is a total code length, consisting of two parts corresponding to the two components just mentioned. This codelength may be considered to be a measure of the complexity of the data with respect to the 2^n-tree model class and the model that produces the lowest complexity (code length) will be sought.

This complexity measure is used in an algorithm where the initial tree is pruned, until the total code length begins to increase. When a branch is pruned from the tree, the code length of the model (tree) decreases and the code length of the class codes conditioned on the model either increases or remains unchanged.

The classification trees produced by this pruning technique are shown experimentally to be both significantly smaller and better at classifying new feature vectors than the unpruned trees.

The contributions of this paper are the following.

1. The application of MDL-based pruning to 2^n-tree classifiers.

2. The specific objective function and associated pruning procedure. This objective function is based on the MDL principle and a tree encoding scheme that utilizes a combinatorial enumeration of 2^n trees.

The objective function obtained is:

$$L(C, \mu|X) \quad = \quad L(C|\mu, X) \ + L(\mu|X) \tag{0.1}$$

Where $L(C|\mu, X)$ is the encoding length of the class codes conditioned on the tree (μ) and the feature vectors (X) and is given by:

$$L(C|\mu, X) \;=\; \log\left[(k_{\max}+1)\binom{k+M-2}{M-2}\binom{N}{k_1, k_2, \ldots, k_{M-1}}\right] \tag{0.2}$$

Also, $L(\mu|X)$ is the codelength of the tree and is given by:

$$L(\mu|X) \;=\; \log\left[\nu_{\max}(2^n, m-1)\;\beta(n_d; 2^n, m-1)\;\binom{n_l+M-1}{M-1}\binom{n_l}{q_1, q_2, \ldots, q_M}\right] \tag{0.3}$$

The various symbols used in these formulas are defined under “notation” below.

1. Outline

2. The Problem

The problem addressed by the work described in this paper is that of "overfitting" of the training data by the 2^n-tree classifier. This occurs when a limited amount of training data is available to train the classifier. This situation is the usual one in practice. Before describing the problem in more detail, the 2^n-tree classifier will be explained. The following description is excerpted (with a few minor changes) from [1, 2]. See those sources for more details.

2.1. The classification tree

In many pattern classification applications (and especially those requiring high throughput) feature values are represented as integers whose range is determined by the wordsize of the computational device being used and is therefore a power of two. Thus the feature space is discrete and occupies a finite n-dimensional hypercube of side length 2^m. The classification tree used in this invention corresponds to the following recursive decomposition of that feature space. Each node in the tree is either a leaf or an interior node (decision node) with 2^n children. The root node of the tree corresponds to the entire feature space and the set of all its children corresponds to the complete feature space divided into 2^n non-overlaping equally-sized hypercubes, each 2^{m-1} on a side and each of their children corresponds to a hypercube of size 2^{m-2} and so on. The tree may be diagrammed in the standard 2-dimensional form as follows. First an arbitrary ordering of the features is selected. This ordering is indicated by the subscripts: $\{x_1, x_2, \ldots\}$ etc. The branches originating at the root node are then labeled by the high order bits of the features. Thus the left most branch would correspond to all feature high-order bits being 0 and the next one to the right would correspond to only the n-th feature having its high-order bit equal bit equal to 1 and so on with the right-most branch corresponding to all high-order bits being 1's. At the next level (level 1) the next to the highest order bits are tested and so on with low order bits being tested at $m-1$ level nodes, all m^{th} level nodes being leaves. For $n = 2$ the tree takes the form of the well-known quadtree data structure.

When the tree is ready to be used for classification (i.e. it has been "trained") the leaves contain class labels. A feature vector is classified by searching down the tree accessing smaller and smaller cells of feature space that contain the feature vector until a leaf is reached. The class label stored at that leaf is then assigned to the vector in question. A simple example is shown in 1. The case shown here is for two features and two classes (i.e. $n = 2$ and $M = 2$).

The tree, as described here, acts as a sort of lookup table (LUT). The advantage of a LUT over other classification schemes is that it allows any subdivision of feature space. The advantage of this representation over a simple LUT is in memory savings. This is important for high dimensional feature spaces because the memory required for a simple LUT is 2^{mn} locations. For 10 features measured with 8 bit resolution a memory size of 2^{80} (or about 10^{24} bytes would be required. In most applications, however, the classes form large connected regions in feature space. Thus with some suitable encoding scheme it should be possible to significantly reduce the memory required to store the information contained in this LUT. Considering only storage requirements, one might be led to a tree of some structure other than the 2^n-tree described above. For example the nodes might correspond to regions bounded by general curvilinear surfaces. This would require significantly more complicated tests to be executed to determine which branch to take from a given node, however, and would reduce the throughput obtainable. As discussed in [1, 2], the 2^n structure results in a hardware architecture that requires almost no overhead for these tests. The simple testing of bits required by this structure can be effected by simply routing the data bits of one stage to the correct address bits the succeeding stage.

2.2. Training

While it is possible, in principle, to implement any classifier using the 2^n-tree scheme, the preferred algorithms are non-parametric procedures that produce an unpruned tree with the property that for any non-training feature vector to be classified one of the following will be true:

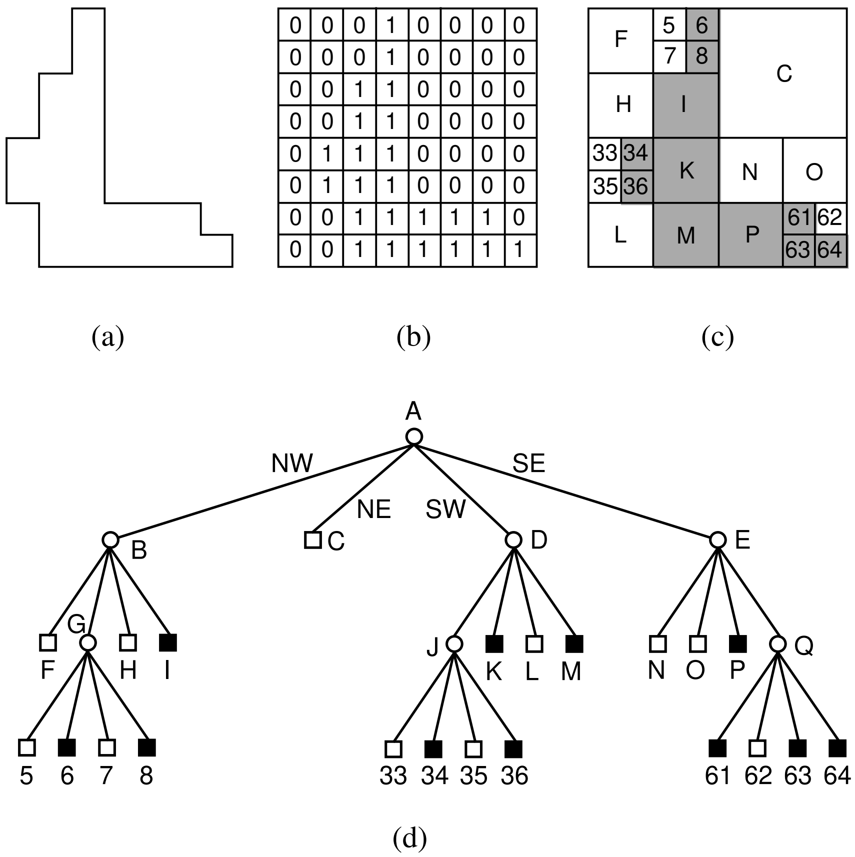


Figure 1: A diagram of a 2^2 (aka. "quad") tree for a two-class two-feature problem. Region (a), the 1's in (b) and the shaded area in (c) correspond to the black leaves on the tree.

1. One or more training vectors with its value were encountered in the training data. In this case the class returned will be the one for which the most training feature vectors with this value were encountered.

2. No training vectors for this value were encountered. In this case the feature vector value is associated with one of the values that was encountered via the tree structure in a nearest-neighbor-like fashion.

2.3. The Pruning problem

As can be seen from this description, the 2^n-tree constitutes a model with a very high number of degrees of freedom For this reason 2^n-tree classifiers are susceptible to fitting-the-noise[1] problems. This is especially true in regions of feature space where there is inherent confusion between classes and the training data are sparse. With this model (the tree), reducing the number of degrees of freedom is equivalent to pruning the tree. This pruning operation is an optimization problem; an

[1] *fitting-the-noise* is also referred to as over fitting and over adaptation. This is the phenomenon wherein a model with too many adjustable parameters adapts to the noise in stochastic data. An example is polynomial curve fitting of noisy data. The Lagrange interpolating polynomial goes through every data point, but in the case of noisy data it will almost certainly do worse at prediction than a lower order polynomial.

overpruned tree is likely to produce worse results than an unpruned one. As with any optimization problem, there must be some *objective function* to be minimized by the optimization process. In this case the desired objective function is the classification error rate on subsequent feature vectors. Unfortunately this is unrealizable almost by definition. The most common approach for addressing this problem is *cross validation*[8, 9], often referred to by names such as "leave-one-out" or "leave-k-out". This very computationally costly procedure attempts to estimate the error by "leaving out" the training vectors one at a time (or k at a time), forming a classifier with the remaining vectors, and then using the resulting classifier to classify the "left-out" vector. This is repeated for all training vectors, computing the average error rate, which is taken to be an estimate of the error rate that would be obtained on subsequent (i.e. not from the training set) feature vectors. In addition to the computational cost of this procedure, it has been shown to be inconsistent[10].

3. The Objective Function

3.1. The Minimum Description Length (MDL) Principle

The philosophical foundation of the minimum description length (MDL) principle is the assertion that the best model for describing a set of data is the one that allows it to be encoded with the shortest length. This may be viewed as a formalization of a principle applied by physicists, known as Ockham's Razor, which states that the simplest model explaining all the observations is the best one.

Code Lengths and Probabilities: Coding theory is concerned with (among other things) optimum (in the sense of minimum code length) encoding of symbols or data. In doing this, variable length codes are frequently used and it is therefore desirable to represent the most frequently occurring symbols with the shortest codes. For a so-called "zero memory" source S[2], generating symbols $\{s_i\}$ with probabilities $\{P(s_i)\}$ it can be shown that the shortest expected code length per symbol is obtained by using code lengths given approximately by $L(s_i) = -\log P(s_i)$ and the value of this expected code length per symbol is given by the source entropy H(S):

$$E\{L(s)\} \quad = \quad H(S) \quad = \quad -\sum_i P(s_i)\log P(s_i) \tag{3.1}$$

This result is known as "Shannon's first theorem"[24]. [3]

Because of these relationships probability models are used to encode data as follows. The model is assumed to have some fixed number of adjustable parameters, denoted by the parameter vector, θ. These are adjusted to give the minimum code length for the data. Then the parameters and the data are encoded. Implicit in this view is that the encoder and decoder both "know" the model

[2]This is a source where the probability of emitting a given symbol is independent of what symbols preceded it.

[3]The details of actually constructing binary codes will be omitted here having no bearing on the theoretical results. In fact, the fundamental implications of any theory relying on such details would have to be viewed suspiciously. The code lengths derived from any probability model will be taken as equal to the negative natural logarithms of the associated probabilities. Of course, for binary codes, this logarithm should be taken to the base two, but once again this is a coding detail. Natural logs will be used here because of their convenient mathematical properties.

form. Thus the code length for a set of data, X, is given by: [4]

$$L(X, \theta) \tag{3.2}$$

where $L(X|\theta)$ is the code length for the data given the model and is given by:

$$L(X|\theta) \quad = \quad -\log P(X|\theta). \tag{3.3}$$

The "conditional" ("|") notation is used because, as we shall see, θ can also be viewed as a random variable. Further justification for relating codes and probabilities this way comes from the well known *Kraft inequality*, which is concerned with prefix codes.[5] For a set of symbols $S \ = \ \{s_1, s_2, \ldots, s_n\}$ with code lengths $\{L(s_i)\}$, the Kraft inequality provides a necessary and sufficient condition for the existence of a prefix code. For binary codes this inequality is:

$$\sum_{i=1}^{n} 2^{-L(s_i)} \quad \leq \quad 1. \tag{3.4}$$

For most reasonably efficient codes the equality holds. Thus for our purposes (which are, after all, not really coding), we can assume that this is true . If we like, we can think of the negative logarithms of the relevant probabilities as fairly tight lower bounds on the actual code lengths. We may similarly write the code length of θ in terms of a probability function.

$$L(\theta) \quad = \quad -\log P(\theta) \tag{3.5}$$

The function $P(\theta)$ reflects prior knowledge about the distribution of θ values and is frequently referred to as simply a *prior*. The meaning and use of this prior are central to the methods of stochastic complexity and will be discussed in more detail below.

The Minimum Description Length (MDL) Principle Equation (3.3) shows that the θ that minimizes the code length $L(X|\theta)$ is the maximum likelihood (ML) estimate. While this minimizes the conditional code length, $L(X|\theta)$, it does not necessarily minimize the total code length, $L(X, \theta)$ (equation 3.2). By choosing θ to minimize this instead, we allow $P(\theta)$ to bias the estimate away from unlikely values of θ and toward likely ones. For very large data sets, X, there will be little difference in the estimates of θ, but this is not true for smaller ones. Choosing θ to minimize the total code length of equation (3.2) is an example of the minimum description length principle (MDL). This may also be viewed as a kind of global maximum likelihood principle, where θ is chosen to maximize the joint probability:

$$P(X, \theta) \quad = \quad P(X|\theta)\ P(\theta), \tag{3.6}$$

rather than the usual conditional probability, $P(X|\theta)$.

A more important case (and the one with which we will be primarily concerned here) is the one in which both the value of θ and its dimensionality, k, may be chosen to minimize the description

[4] The symbol X is used here to represent the entire set of data being considered. It may represent a string of symbols to be encoded, a set of data to be encoded (or fit with a model) or a set of pattern classifications.

[5] Prefix codes (also known as *instantaneous*) are codes where no symbol's code is a prefix for another. With codes of this type there is no need for a symbol separator or break character.

length. We will refer to $P(X|\theta)$ and $P(\theta|k)$, for a given k, as a *model* and the set of all such models (corresponding to all k values) as a *model class*, M, whose complete description is specified by: $\{P(X|\theta), P(\theta|k), P(k)\}$. In this case, if we follow the procedure outlined above for encoding using probability models, the code length we are concerned with minimizing is:

$$L(X,\theta,k) \;=\; L(X|\theta) \;+\; L(\theta|k) \;+\; L(k) \tag{3.7}$$

which is equivalent to maximizing the joint probability:

$$P(X,\theta,k) \;=\; P(X|\theta)P(\theta|k)P(k) \tag{3.8}$$

Once again, this can be viewed as a maximum likelihood problem, where estimates $(\hat{\theta},\hat{k})$ for (θ,k) are chosen to maximize this joint probability.

As we shall see below, this will have application (for example) in the problem of dimensionality determination, where we wish to determine the optimum k value for reasons of classifier performance and computational cost.

As an example, consider the problem of curve fitting or regression analysis and to make things concrete consider the case of fitting polynomials to data of the form $X = \{(x_i, y_i), i = 1, 2 \ldots N\}$ where x is the independent and y is the dependent variable. In this case we are interested in modeling the behavior of the dependent variable given the set of values of the independent variable. One can always obtain increasingly better fits (as measured by the mean square deviation from the regression curve) by going to higher and higher order polynomials until the Lagrange interpolating polynomial (which goes through every point) is obtained and the error becomes zero. It is well known (and intuitively obvious) that beyond a certain order, however, the curves will become worse at predicting future values of y, given the corresponding x values. Also, if the parameters are to be used to determine a physical model, its determination will also become less reliable beyond a certain order.

This problem of determining the optimum order is ideally suited for the application of stochastic complexity. To use our coding analogy, we are interested in the case where the $\{x_i\}$ are known to both the encoder and the decoder and we wish to encode the $\{y_i\}$. To put the problem on a probabilistic footing, we will assume the Gaussian distribution commonly used in such problems. Thus we are interested in maximizing:

$$P(\{y_i\},\theta,k|\{x_i\}) \;=\; P(k)\; P(\theta|k)\; (2\pi)^{-N/2}\sigma^{-N} \times \exp\left[\frac{-1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i \;-\; \sum_{j=0}^{k-1}\alpha_j x_i^j\right)^2\right] \tag{3.9}$$

where $\theta = (\{\alpha_j\},\sigma)$. In this case it can be shown that the maximum likelihood estimates for the $\{\alpha_j\}$ are the least squares estimates. Table 1 shows a sample result from using stochastic complexity to determine the proper order for the case where a set $\{(x_i, y_i)\}$ were generated using $y = x(x-5)(x-7)(x-10)$ with Gaussian noise of mean zero and $\sigma = 10$ added to the y values. Here the correct answer was chosen as the one with the minimum description length.[6] The prior

[6]For computational convenience a number that varies monotonically with the real description length was calculated.

Table 1: Example of using stochastic complexity to determine the optimal polynomial in curve fitting.

k	TERMS	VARIANCE OF FIT	"DESCRIPTION LENGTH"
1	0	7620	1.50
2	0 1	4900	1.47
3	0 2 3	3610	1.44
4	1 2 3 4	76.8	0.978
5	1 2 3 4 5	73.8	0.989
6	0 1 2 3 4 5	72.7	1.00

probability of having k parameters, $P(k)$, was assumed the same for all k so it was ignored, and the form used for $P(\theta|k)$ was that derived by Rissanen in [25]:

$$P(\theta|k) \quad \sim \quad N^{-k/2}, \tag{3.10}$$

(This was derived for the case of k parameters computed from N data observations with no "real" prior knowledge. At each k, all possible sets of terms with k elements, up to 5th order terms, were checked. The column labelled "TERMS" in the table shows the specific terms selected (0 the constant term, 1 the linear term, etc.). The data and some of the curves corresponding to these results are plotted in Figure 2. To avoid the undesirable aspects of cross validation, we will

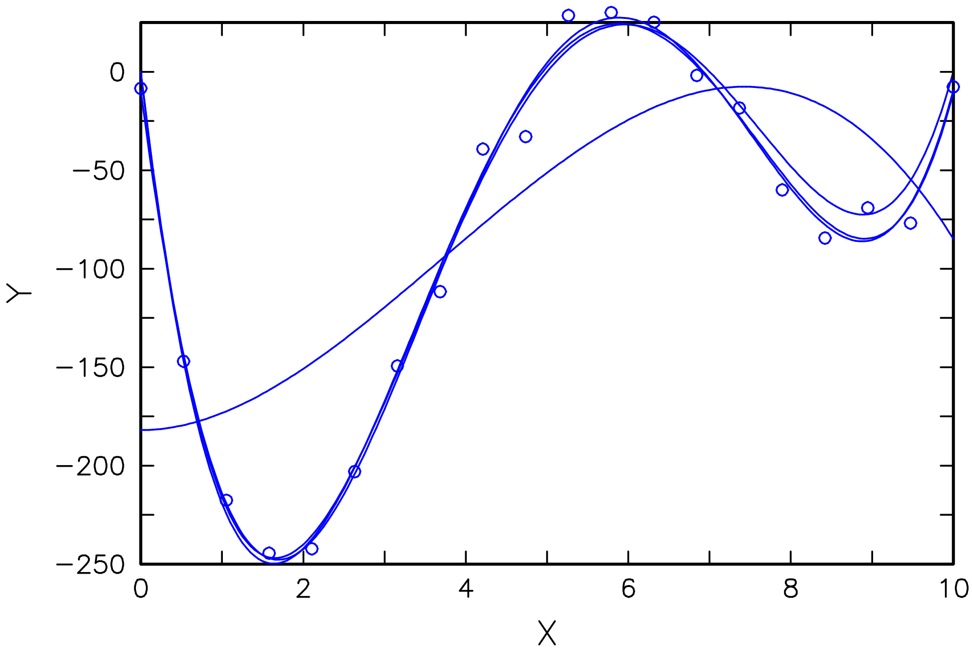


Figure 2: Results from applying MDL to curve fitting. The bold line is a superposition of the generating function and the 4-parameter fit chosen using MDL. The other two curves are the selected 3 and 6 parameter fits.

follow a scheme similar to that of Quinlan and Rivest[5] but with important differences. We will apply Rissanen's Minimum Description Length (MDL) principle[7] as follows to obtain an objective function to be minimized by the pruning procedure. Classification results obtained using Gaussian classifiers obtained by doing MDL-based feature selection were shown to produce results superior to those obtained using cross validation for feature selection[17]. We assume a data encoding/decoding problem where the encoder and decoder both have access to the feature vectors describing a set of objects, but only the encoder has access to their class codes. Our task is to construct an encoding scheme based on the 2^n-tree that minimizes the codelength of the class codes given (i.e. conditioned on) the feature measurements.

The so-constructed code will have two parts, each with its associated length. There will be the cost (in bits) of encoding the tree and the cost of encoding the class codes of those objects that the tree misclassifies.[7]

$$L(C,\mu|X) \;=\; L(C|\mu,X) \;+\; L(\mu|X) \tag{3.11}$$

Here μ represents the model, which in our case is the tree, and $(C|\mu,X)$ is the description of C "conditioned on" μ and X. This is the codelength for the exceptions or misclassified training vectors.

3.2. Coding the Exceptions

3.3. Two-Class Problems

To encode the exceptions (those vectors incorrectly classified by the tree) we will first propose a description and then a scheme for compressing it. For the case of two-class problems, this is the same scheme used in [5]. For two-class problems the misclassifications are identified by a binary string with one binary digit corresponding in order to every training vector. If a vector was classified correctly by the tree, its digit is set to zero and is set to one if the vector was misclassified. Because, in our coding metaphor, the decoder will have the classification tree, it knows simply to assign the class not assigned by the tree. We will refer to this string as s. In most applications the total number of misclassified training vectors, k, will be much less than the total number of training vectors, N. We can utilize this fact in the following compression scheme, proposed in [5].

The idea is to transmit a code for k and a code for s conditioned on k, $(s|k)$. [8] We will encode k as a fixed-length binary integer of $\log(k_{\max}+1)$ bits,[9] where[10] $k_{\max}$ is the maximum possible value of k. The binary string will be compressed as follows. there are $\binom{N}{k}$ binary strings of length N with k 1's. Thus to encode these strings we will simply enumerate them. This gives the following

[7] The correct class codes don't need to be encoded because the decoder has access to both the feature vectors and the classification tree.

[8] We will really transmit a code for s conditioned on both k and N, but we do not mention N explicitly, because, in our coding metaphor, it is already known to the decoder, which has free access to the feature vectors in the training set.

[9] All logarithms in the paper are base-two unless otherwise stated.

[10] If we were to actually use the scheme to encode values, we would, of course, use the ceiling of this quantity. Since the encoding metaphor is simply that, we will not concern ourselves with such details. If anything a case can probably be made on fundamental grounds that it's better to use the log than its ceiling.

total code length for s.

$$L(s) \;=\; L(N, k, k_{\max}) \;=\; \log(k_{\max}+1) \;+\; \log\binom{N}{k} \tag{3.12}$$

This has the following probability interpretation, based on MDL. Minimizing the codelength we are minimizing is equivalent to minimizing:

$$-\log P(C, \mu | X) \;=\; -\log P(C|X, \mu) \;-\; \log P(\mu|X) \tag{3.13}$$

where μ is a somewhat abstract variable identifying the model. In our case it can be made more concrete because the number of possible 2^n-ary trees of maximum depth m is clearly finite and enumerable. We may consider μ to simply be an integer index that identifies the tree. Because of the nature of our model, X tells us very little about μ without C. Therefore we may assume that $P(\mu|X) = P(\mu)$. Using this interpretation, equation (3.12) implies that we assume that all k values are equally probable and that for a given k, all distributions of those misclassifications within the training data are equally likely. Unless we have a meaningful ordering of the training data, the latter assumption is a natural one. The former is probably not, however.[11]

3.4. More than Two Classes

One way to handle the case of more than two classes is to use the binary scheme just described and then to follow s with a list of the correct class codes corresponding to the misclassified training vectors (i.e. the 1's in s). Each of these can be coded using fixed-length codes of $\log(M-1)$ bits each.[12] Thus the total codelength for the exceptions is given by

$$L(s) \;=\; \log(k_{\max}+1) \;+\; \log\binom{N}{k} \;+\; k\log(M-1) \tag{3.14}$$

In cases where the number of exceptions is sufficiently large further compression may be obtained by using variable-length class codes based on the frequency of occurrence of certain classes or (since the default tree class comes with the tree "for free") codes whose length is given by $-\log P(c_r|c_t)$ where c_r is the real or correct class and c_t is the class produced by the tree classifier. There is, of course, an "overhead" associated with using such codes because the associated "codebook" must be transmitted to the decoder.

Assuming that $M \ll k$, which will usually be the case in applications of interest to us, there will be a lot of redundancy in the just-described list of class codes. This can be utilized in the following scheme, which further compresses the description. Rather than simply listing the class codes of the misclassified vectors, we will divide that list into two pieces. The first will be a vector of counts of the number of times each class label appears in the string. We will denote this by the set: $\kappa \equiv \{k_1, k_2, \ldots, k_{M-1}\}$. Then we will encode the list of class assignments by enumerating all

[11] Note that this interpretation favors k's that are near zero or N. Usually $k \ll N$. Using the straight-forward N-bit binary string s is equivalent to assuming that all possible strings are equally probable.

[12] This codelength is a multiple of $\log(M-1)$ rather than $\log M$ because the default class that would be returned by the tree is known to the decoder and can thus be eliminated.

possible labelings consistent with κ. The number of these is given by the multi-nomial coefficient:

$$\binom{k}{k_1, k_2, \ldots, k_{M-1}} \tag{3.15}$$

The list of class counts, κ, can be further compressed by enumerating all possible κ's consistent with k. The number of these is given by:

$$\binom{k+M-2}{M-2} \tag{3.16}$$

This expression is derived in Appendix B.

Summarizing, our code for the misclassified vectors consists of the following components:

1. A code for the total number of misclassifications, k. This requires $\log(k_{\max}+1)$ bits, where $k_{\max} = (N+1)/2$ is a reasonable choice.
2. A code for the misclassification counts by class label, κ. This requires $\log\binom{k+M-2}{M-2}$ bits.
3. A code for the binary string that identifies which training vectors were misclassified. This requires $\log\binom{N}{k}$ bits.
4. A code for the list of class codes to be associated with the above-mentioned binary string. This requires $\log\binom{k}{k_1, k_2, \ldots, k_{M-1}}$ bits.

This expression may be simplified and written in the form[13]:

$$L(C|\mu, X) = \log\left[(k_{\max}+1)\binom{k+M-2}{M-2}\binom{N}{k_1, k_2, \ldots, k_{M-1}}\right] \tag{3.17}$$

It should be pointed out that N, k and the $\{k_i\}$ are usually large enough that Stirling's approximation can be used to compute most of the logarithms (e.g. equation (3.17)) required to compute the length for encoding the exceptions.

3.5. Coding the Tree

3.6. The Trees and Scheme of Quinlan and Rivest

The problems analyzed by Quinlan and Rivest[5] were the following. The trees were of unlimited depth and variable arity. Also the attribute to be tested at a given node had to be included in the tree description. They were also concerned primarily with attributes (i.e. features) with a range consisting of a small discrete number of values (usually two). They mention the possibility of real-valued features, but only discuss the case where the description of the "attribute" will include the feature to be tested as well as the value of a "cut point" or threshold. This is for describing tests like "$x > 98.6$?", for example. In [5] these trees are encoded using the following recursive scheme that results in the nodes being stored in *preorder*.

1. 1 bit identifying this node as a decision node or a leaf

[13] Equation (10) on p.237 of [5] corresponds to this, but that equation appears to be in error.

2. the default class if the node is a leaf (1 bit in their case)

3. the attribute to be tested if the node is a decision node.

4. if the node is a decision node, the description of all the subtrees corresponding to the node's children (i.e. the child is the root) in order, from left to right.

They point out that this scheme is nearly optimal for binary trees, but not for trees of higher arity. Unfortunately, the possibility of variable arity makes it difficult to come up with a compression scheme.

3.7. A Scheme for 2^n-Trees

In contrast to the trees studied in [5], 2^n-trees have the following properties:

1. fixed arity $= 2^n$.

2. a maximum depth limitation of m.

3. the test to be carried out at a decision node is determined by the position of that node within the tree. Thus there is no need to code the attribute for decision nodes.

These facts, combined with the fact that in the applications of interest to us, the number of leaves is very much larger than the number of classes, M, admit a significant compression of the description of the tree. Our description of the tree will be divided into two parts. The first is a structural description of the tree and the second provides the class labels to be assigned to the leaves.

Let $n_T = n_d + n_l$ equal the total number of nodes in the tree, where n_d is the number of decision nodes and n_l is the number of leaves. It can be shown that

$$2^n n_d = n_d + n_l - 1 \tag{3.18}$$

Thus there is a fixed relationship between n_d and n_l independent of the details of the tree structure. For this reason we will condition the tree structure description on n_d.

Let $\beta(\nu; a, h) =$ the number of ordered trees with ν nodes, arity a and depth not greater than h. Here we mean trees that aren't necessarily fully populated in the sense that a decision node (or non-leaf) can have from 1 to a children. By *ordered* we mean that there is significance to the position of a child. Thus there are a possible two-node trees. Such a tree is formed by the decision nodes of a 2^n-tree. Thus we can encode the structural description of a 2^n-tree with $\log[\beta(n_d; 2^n, m-1)]$ bits. We also must encode n_d, of course, which will require $\log[\nu_{\max}(2^n, m-1)]$, where $\nu_{\max}(a, h)$ is the number of nodes in the maximal a-ary tree of depth h.

$$\nu_{\max}(a, h) = \sum_{l=0}^{h} a^l = \frac{a^{h+1} - 1}{a - 1} \tag{3.19}$$

Because this part of the codelength is fixed for a given problem, however, we will ignore it during the pruning process.

At the time of this writing, it appears that there is no known closed form expression for $\beta(\nu; a, h)$[19, 20]. It can be computed exactly using a recursive scheme and asymptotic forms may be constructed for various ranges of the parameters. These computational methods are discussed in Appendix C. A closed form for the counting function for trees of unlimited depth does exist and is given by:

$$\beta(\nu; a, \infty) = \frac{1}{(a-1)\nu + 1}\binom{a\nu}{\nu} \tag{3.20}$$

This expression is derived in Appendix C and its log may constitute an adequate approximation to $\log[\beta(\nu; a, h)]$ in some cases.

In addition to the tree structure, we must also describe the labeling of the leaves. The expression for this encoding length is very similar to that of equation (3.17):

$$\log\left[\binom{n_l + M - 1}{M - 1}\binom{n_l}{q_1, q_2, \ldots, q_M}\right] \tag{3.21}$$

Where q_i = the number of leaves with class label i. Note that:

$$\sum_{i=1}^{M} q_i = n_l \tag{3.22}$$

The description need not include n_l because is can be computed from n_d using (3.18). We may now combine these expressions to get a single expression for the encoding length of the tree:

$$L(\mu) = \log\left[\nu_{\max}(2^n, m-1)\ \beta(n_d; 2^n, m-1)\ \binom{n_l + M - 1}{M - 1}\binom{n_l}{q_1, q_2, \ldots, q_M}\right] \tag{3.23}$$

Equation (3.23) combined with (3.17) give us an expression for our objective function (3.11). Next we will describe the algorithm for applying this function to the problem of pruning 2^n-tree classifiers.

3.8. A Comment on the Coding Scheme for 2^n-Trees

The coding scheme we have chosen is based on and takes advantage of the structure of 2^n-trees. This seems like a good thing because it allows 2^n-trees to be coded with shorter code lengths than schemes that don't take as much advantage of their structure. There is a counter point to this reasoning, however. The 2^n tree was chosen, not because it is a natural representation for the training data, but rather because it is convienent for hardware and software implementation. Before explaining this point, we will take a brief digression on the relationship between MDL and the Bayesian maximum likelihood principle (ML).

Think of the model, μ, as a random variable like X We will think of the system that generated our training data (the one corresponding to μ) as having been chosen from a population or ensemble of systems characterized by a probability density $P(\mu)$, normally refered to as the *prior*. In this conceptual scenario the selection of the training set, $\{C, X\}$, is a two-step process consisting of first selecting the generating system from the population corresponding to $P(\mu)$ and then selecting the training set from the population corresponding to μ and described by $P(C, X|\mu)$. Our estimation problem will then consist of choosing the model, $\hat{\mu}$ that maximizes $P(\mu|C, X)$, which is equivalent to maximizing $P(C, X, \mu)$. Note that:

$$P(C, X, \mu) = P(C, X|\mu)P(\mu) \tag{3.24}$$

and

$$-\log P(C, X, \mu) \;=\; -\log P(C, X|\mu) - \log P(\mu) \tag{3.25}$$

Thus we see that in this scenario, where we know $P(\mu)$, this ML estimate and the MDL estimate are identically equivalent. What about the case where we don't know $P(\mu)$ or (depending on one's philosophical position) it may be questionable whether or not it is even meaningful to talk about the existence of such a "real" prior? We believe that it is meaningful to talk about this real prior to the following extent. If we know something about the mechanism (e.g. the physical phenomena) that generated our training data, we may be able to specify certain reasonableness constraints on $P(\mu)$. For example, we don't want it to be strongly peaked around certain arbitrary (with respect to the data) values of μ.

The reason for this digression is to point out the following. Due to the structure of 2^n trees, certain trees will be favored over others because of their simplicity, where there simplicity is, to a large degree, only meaningful within the context of 2^n trees and not in any significant way with respect to the problem or the data.

As an example consider a two-class two-feature problem, where we wish to compare the following to decision boundaries. In case A. the decision boundary is a vertical line bisecting the horizontal (call it feature one) axis. This has a quite simple representation - a five-node tree with a root node and four leaves - two class-one and two class-two. In case B. we wish to move the vertical decision boundary of case A. one feature resolution unit to the right (i.e. increase it by one). By any kind of reasonableness consideration, we would say that this representation is about as complex as that of case A. and should be equally favored. The case B. tree has $5 + 8(m-1)$ nodes for m-bit feature encoding that would mean 61 nodes for 8-bit features, for example.

4. The Pruning Algorithm

Now that we have a measure of the complexity of a given tree representation of the training data, we must construct an algorithm to find such a description with suitably low complexity. We would like to find the tree with the lowest complexity, but previous theoretical results[18] suggest that this problem may be np-complete, forcing us to accept a sub-optimal solution for practical reasons.

In [5] the complexity measure is used both in growing and in pruning. Because of the nature of 2^n-trees and the way they are grown[1, 2], however, it is not relevant to the growing phase. Thus we will use it only in the pruning phase. The algorithm, described in simple terms, consists of pruning the tree in a bottom-up, breadth-first traversal, where leaves are pruned if doing so produces a reduction in the complexity of the tree-based description of the training data. That is the cost of encoding the increased number of misclassifications (of the training data) is at least offset by the reduction in the code length of the tree. Starting with the next-to-the-bottom-level nodes, all decision nodes are examined to see if converting them to leaves would reduce the code length (complexity). When a decision node is converted to a leaf, all its children are pruned and the class assigned to it is the one for which most training vectors in its quad were encountered.

The computational complexity is roughly $O(n_T)$. Every node is visited once during the recursive traversal of the tree, but the objective function is only evaluated at those nodes where pruning stops and at their descendants.

5. Experimental Results

Experiments were carried on both synthetic and real data. In these experiments, due to the difficulty in computing $\beta(\nu; a, h)$, $\beta(\nu; a, \infty)$ was used as an approximation. Also, Stirling's approximation was used where appropriate. Despite these approximations, quite good results were obtained.

5.1. Classifying Synthetic Gaussian Data

A sample of 2000 vectors with two classes distributed as a pair of overlapping Gaussian regions in a two-dimensional feature space was used to train the tree classifier. Error rates were calculated by classifying 80,000 new vectors from the same distributions. Table 2 shows the tree structure, size and classification error rates before and after pruning. A scatter plot of the training data and the feature-space decision regions corresponding to the unpruned and pruned trees are shown in Figure 3.

Table 2: Tree sizes and error rates obtained before and after pruning. These data are for Gaussian synthetic data.

Tree Size Information						
	Before Pruning			After Pruning		
Depth	Internal Nodes	Leaves	Word Count	Internal Nodes	Leaves	Word Count
1	1	0	0	1	0	1
2	4	0	4	3	0	4
3	9	0	16	4	1	12
4	15	2	36	8	1	16
5	33	5	60	9	3	32
6	76	10	132	6	6	36
7	169	23	304	1	8	24
8	0	338	676	0	2	4
Totals	307	378	1228	32	21	129

Error Rates

without pruning:	22.0 %
with pruning:	17.0 %
Bayes Rate:	15.2 %

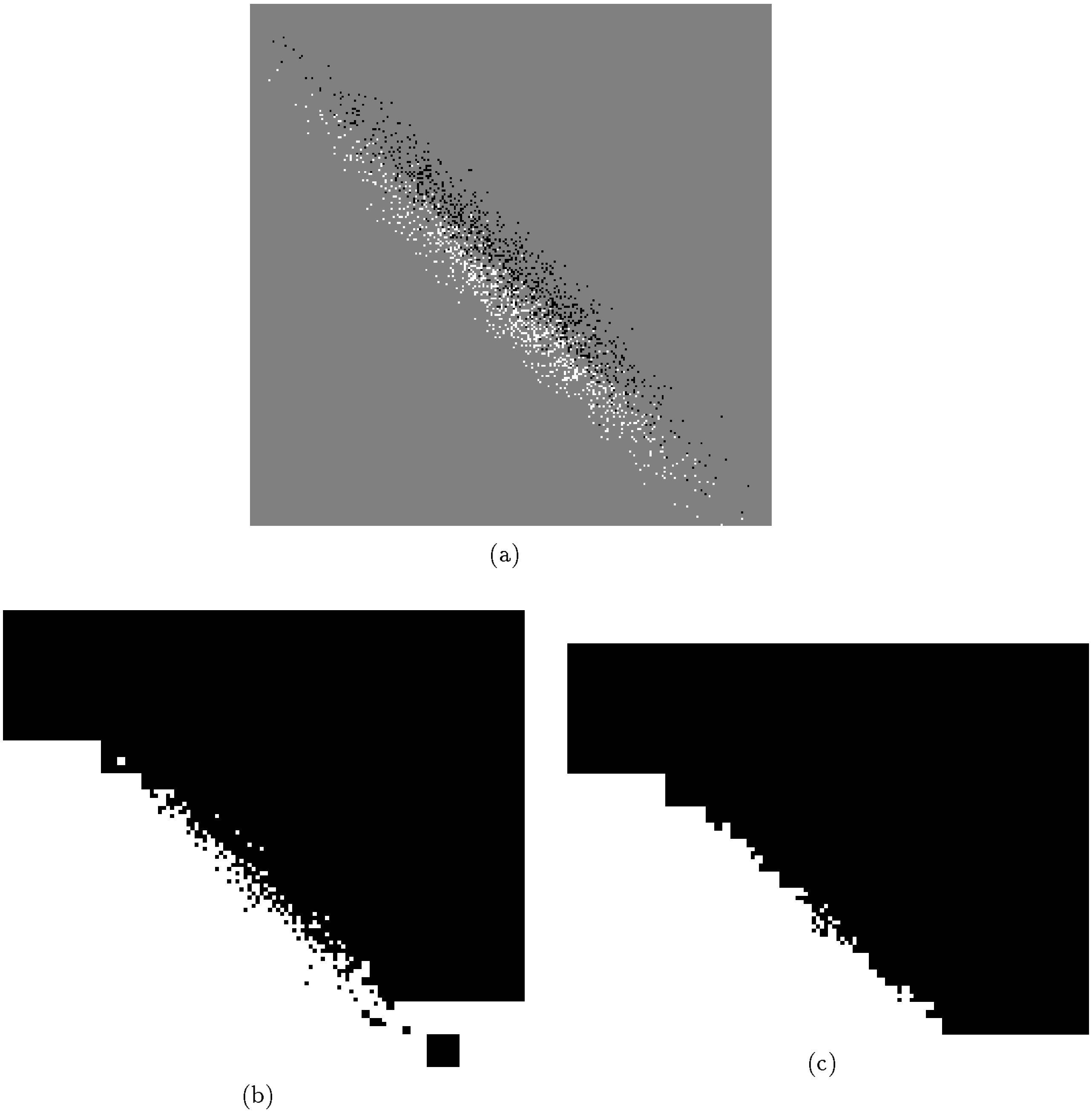


Figure 3: (a) Scatter plot corresponding to the synthetic data used in some of the pruning experiments. Overlapping Gaussians were used to generate 2000 vectors; (b) Feature-space decision regions corresponding to the unpruned tree corresponding to this training data; (c) The corresponding tree after pruning.

5.2. Segmentation of Thin-Film Images

The technique described here has also been used in constructing classifiers for segmenting multi-band images of thin-film wiring used on high-end first-level electronic packages. In this experiment the unpruned tree contained a total of 17,720 nodes, while the tree resulting from MDL pruning was only 1124 nodes and produced a lower classification error rate. In fact, the cost-weighted error rate obtained on new images (not the training data) for the pruned tree was half that of the unpruned tree.[14]

6. Other Related Work (Rissanen and Wax)

The only previous applications of MDL to the construction of decision trees of which we are aware are the work of Quinlan and Rivest[5] (described above) and that of Rissanen and Wax[3] (also described in [4]). The latter treats the problem of constructing binary decision trees where both growing and pruning decisions must be made. The pruning decision is made as the tree is being grown. That is, a decision is made as to whether or not to split a given node thus creating its children or to terminate that path in the tree by making the node a leaf. The other choices to be made are: which feature to base the split on (only one is tested at a given node) and the associated threshold value. In this approach the pruning decision is made locally based only on the description length for the training samples corresponding to a given node. The description length of the tree is not considered.

7. Conclusion

We have developed and evaluated a scheme for pruning 2^n-tree classifiers. This scheme is based on the minimum description length principle. The trees produced are significantly better at classification than unpruned trees and an order of magnitude smaller.

References.

[1] B. Dom and D. Steele, "2^n-Tree Classifiers", *IBM Invention Disclosure No. SA890-0042*, Submitted 1/30/90; Rated "publish" 3/90.

[2] "2^n-Tree Classifiers and Realtime Image Segmentation", B.Dom and D.Steele, (IBM Research Report RJ 7558 (70424)) (7/2/90)

[3] J. Rissanen and M. Wax, "Algorithm for Constructing Tree Structured Classifiers", *U.S. Patent No. 4,719,571*, 1/12/88

[4] M. Wax, "Construction of Tree Structured Classifiers by the MDL Principle", *Proc ICASP*, (1990) 2157-2160

[5] J.R. Quinlan and R.L. Rivest, "Inferring Decision Trees Using the Minimum Description Length Principle", *Information and Computation*, **80**, (1989) 227-248

[14] The associated figure-of-merit numbers were 2235 and 1166.

[6] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, Vol. 15 in the World Scientific Series in Computer Science, ISBN 9971-50-859-1 (1989),

[7] J. Rissanen, "Modeling by Shortest Data Description", *Automatica*, **14**, (1978) 465-471

[8] H. L. Gray and W. R. Schucany, *The Generalized Jackknife Statistic*, Marcel Decker, 1972.

[9] F. Mosteller and J. W. Tukey, "Data Analysis Including Statistics" in *The Handbook of Social Psychology*, edited by G. Lindzey and E. Aronson, Addison-Wesley, 1968.

[10] See [6] pp.95-96.

[11] Classification and Regression Trees by L. Brieman, J. Friedman, R. Olshen and C. Stone (1984) PUBL: Wadsworth International group (Belmont, CA)

[12] P. Chou, "Applications of Information Theory to Pattern Recognition and the Design of Decision Trees and Trellises", *Ph.D. Thesis, Stanford University* (1988)

[13] P.A. Chou, T. Lookabaugh and R.M. Gray, "Optimal Pruning with Applications to Tree-Structured Source Coding and Modeling", *IEEE Trans Inf Th*,**35**(2), (3/89) 299-315

[14] P. Chou, "Optimal Partitioning for Classification and Regression Trees", *IEEE Trans PAMI*,**13**,(4) April '91, 340-354

[15] J.R. Quinlan, "Induction of Decision Trees", *Machine Learning*,**1**, (1986) 81-106

[16] S.B. Gelfand, C.S. Ravishankar, E.J. Delp and S.B. Gelfand, "An Iterative Growing and Pruning Algorithm for Classification Tree Design", *IEEE Trans PAMI* (to appear)

[17] "Feature Selection with Stochastic Complexity", (B.Dom, W.Niblack and J.Sheinvald), Proc. 1989 IEEE conf on Computer Vision and Pattern Recognition, pp.241-248

[18] L. Hyafil and R.L. Rivest, "Constructing Optimal Binary Decision Trees is np-Complete", *Information Processing Letters*, **5**(1) (1976) 15-17

[19] A.M. Odlyzko, "Some New Methods and Results in Tree Enumeration", *CONGRESSUS NUMERANTIUM*,**42** (1984) 27-52

[20] A.M. Odlyzko, Private Communication 3/13/90

[21] R.L. Graham, D.E. Knuth and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, 1989, ISBN0-201-14236-8

[22] P. Flajolet and A.M. Odlyzko, "Limit Distributions for Coefficients of Iterates of Polynomials with Applications to Combinatorial Enumerations", *Math Proc Camb Phil Soc*, **96** (1984) pp.237-253

[23] D.E. Knuth, *The Art of Computer Programming; Volume 1: Fundamental Algorithms*, Addison-Wesley, (1973), ISBN 0-201-03809-9

[24] C. E. Shannon, "A Mathematical Theory of Communication", *Bell Syst Tech J.*, **3** pp.379-423 (1948)

[25] J. Rissanen, "A Universal Prior for Integers and Estimation by Minimum Description Length", *The Annals of Statistics*, **2** (11), pp.211-222 (1983)

A. Notation and Terminology

1. x_j = the jth *feature* (or feature measurement). Each of these measurements defines a dimension or axis of the *feature space* relevant to a given classification problem.

2. m = the number of bits of precision used to encode the feature measurements ($\{x_j\}$). There are thus 2^m levels for each feature. Also, the maximum tree depth is m.

3. $x = \{x_1, x_2, \ldots\}$ = *feature vector* = a set of feature measurements corresponding to an object to be classified.

4. n = the number of components of x (i.e. the number of feature measurements). This is also the n in the 2^n-tree classifier, and the number of children per node in the classification trees described here is 2^n.

5. $X = \{x^{(1)}, x^{(2)}, \ldots\}$ = a *training set*, which is a collection of training vectors (the $x^{(j)}$) to be used to train a classifier.

6. N = the number of vectors in X.

7. c = class of the object corresponding to x.

8. M = the number of classes (possible values of c).

9. $C = \{c_j\}$ = a *class assignment vector* whose components are the class assignments of the objects corresponding to the elements (training vectors) of X.

10. $\gamma(x)$ = a classification function (or just *classifier*) for a feature vector, x, the estimated class of the corresponding object is returned.

11. $d_c(x)$ = the *discriminant function* for class c.

12. μ = the "model" used to describe the training data. In our case this will be the decision tree.

13. s = an N-symbol string describing the classes of the training vectors "conditioned on" the classifier. In this string 0's represent correctly classified vectors and for those misclassified, the correct code appears. For two-class problems s is a binary string.

14. k = the number of misclassified training vectors. This is the number misclassified when the classifier constructed during a training process is used to classify the training data. For two-class problems k is the number of 1's in s.

15. number of nodes in a tree:

    (a) n_T = the total number of nodes

    (b) n_d = the number of decision nodes

    (c) n_l = the number of leaves

16. q_i = the number of leaves with class label i.

17. $\beta(\sigma, \nu)$ = the number of strings of ν non-negative integers that sum to σ.

18. $G(x, \nu)$ = a *generating function* for $\beta(\sigma, \nu)$. $G(x, \nu) \equiv \sum_{\sigma=0}^{\infty} \beta(\sigma, \nu)\, x^{\sigma}$. In this usage, x, is a dummy scalar variable and not a feature vector.

19. $\beta(\nu; a, h)$ = the number of ordered trees with ν nodes, arity a and depth not greater than h.

20. $G(x; a, h)$ = *generating function* associated with $\beta(\nu; a, h)$. $G(x; a, h) \equiv \sum_{\sigma=0}^{\infty} \beta(\sigma; a, h)\, x^{\sigma}$. In this usage, x, is a dummy scalar variable and not a feature vector.

21. $\nu_{\max}(a, h)$ = the number of nodes in a maximal a-ary tree of depth h.

22. $\Lambda = \{\lambda_i\}$ = a vector of the node counts of the trees attached to each of the edges originating at the root node of an ordered tree.

23. $||\Lambda|| \equiv \sum_i \lambda_i$ = the *norm* of Λ.

24. $H(x, c)$ = *joint histogram* for class c and feature vector x.

25. $P(x, c)$ = the *joint* probability (or probability density) for x and c. Other standard notation for *conditional* ($P(x|c)$) and *marginal* ($P(x)$) probabilities will also be used.

26. *quad* = a term referring to the region of feature space corresponding to a given node. The higher the level (closer to the root) of the node the larger the region.

27. *arity* = the branching factor of a node. That is the number of children. In a 2^n-tree the arity is 2^n.

28. *preorder* = an order for traversing the nodes of a tree that is a top-down, left-to-right, recursive traversal. That is to traverse a tree one first visits the root and then does a preorder traversal of the left most child's subtree and so on.

B. Enumerating Strings

Our purpose in this appendix is to derive and discuss some combinatorial formulas for enumerating and encoding strings of symbols.

B.1. Binary Strings

We are concerned here with encoding N-digit binary strings with k 1's. If, in our application, it is true that all strings are equally likely, then we have no better coded representation than the standard string of binary digits. For this case we have:

$$\forall_s \;\; P(s) \;=\; 2^{-N} \tag{B.1}$$

From Shannon we know that, for this case, the minimum expected code length (over our ensemble of strings) will be obtained by codes whose lengths are:

$$L(s) \;\;=\;\; -\log P(s) \;\;=\;\; N \tag{B.2}$$

In the scheme suggested in the text of the paper we encode k and then s conditioned on k, $(s|k)$. We encode k with a fixed number of bits and $(s|k)$ with a variable number:

$$L(s) \;\;=\;\; L(N, k, k_{\max}) \;\;\equiv\;\; \log(k_{\max}+1) \;+\; \log\binom{N}{k} \tag{B.3}$$

Where the number of N-bit strings with k 1's is given by $\binom{N}{k}$. This form favors strings for which there a few strings with the same number of 1's. This means that strings with $k \ll N$ and $k \approx N$ will be favored (i.e. have shorter codelengths). In applications of interest to us, we will usually be in the $k \ll N$ regiem.

B.2. More than two symbols

In the text above we handled the case of more than two symbols by differentiating between the "correctly classified" symbol and the other class labels. Specifically, we kept the binary string that indicated which vectors had been misclassified and then we added the list of labels for the misclassified vectors. In it's crudest form this gave a code length of:

$$L(s) \;\;=\;\; \log(k_{\max}+1) \;+\; \log\binom{N}{k} \;+\; k\log(M-1) \tag{B.4}$$

where M is the number of classes and $M-1$ appears in the log because one of the classes can always be eliminated, namely, the one that the tree picked.

Once again, this is similar to assuming that all possible labelings of the misclassified vectors are equally likely. Rather than simply having a list of class codes, we may encode the list by first listing the set of class counts: $\kappa \;\equiv\; \{k_1, k_2, \ldots, k_{M-1}\}$ and then a code for the list of labelings conditioned on κ. The number of k-symbol strings composed of members of a set of $M-1$ symbols and consistent with κ is:

$$\binom{k}{k_1, k_2, \ldots, k_{M-1}} \tag{B.5}$$

This is a standard counting formula that is derived by considering all $k!$ permutations of the k symbols and dividing out all those that are redundant.

We may also compress κ from the straightforward $(M-1)\log N$ bit representation, by first encoding k and then $(\kappa|k)$. The cost of encoding $(\kappa|k)$ is the log of the number of sets of $M-1$ non-negative integers that sum to k. This counting formula is given by:

$$\binom{k+M-2}{M-2} \tag{B.6}$$

This formula may be derived as follows: Let $\beta(\sigma,\nu)$ equal the number of strings of ν. non-negative integers that sum to σ. It can be seen that this obeys the following recursion relation:

$$\beta(\sigma,\nu) \quad = \quad \sum_{l=0}^{\sigma} \beta(l,\nu-1) \tag{B.7}$$

This expression comes from considering each value of the first integer in the set and counting the number of sets of $\nu-1$ integers (the remaining ones) consistent with that value for the first. Applying (B.7) repeatedly yields:

$$\beta(\sigma,\nu) \quad = \quad \sum_{l_1=0}^{\sigma} \sum_{l_2=0}^{l_1} \cdots \sum_{l_{\nu-1}=0}^{l_{\nu-2}} \beta(l_{\nu-1},1) \tag{B.8}$$

where: $\forall_\sigma \; \beta(\sigma,1)=1$.

To find a closed-form for β we will use the technique of *generating functions* commonly used in combinatorial analysis. That is, we define the function $G(x,\nu)$ where:

$$G(x,\nu) \quad \equiv \quad \sum_{\sigma=0}^{\infty} \beta(\sigma,\nu)\, x^\sigma \tag{B.9}$$

This function is used as simply an algebraic device, where x is a dummy variable and there is no concern about whether the series in (B.9) converges. Substituting (B.7) for β, regrouping and collecting terms yields:

$$G(x,\nu) \quad = \quad G(x,\nu-1)\sum_{\sigma=0}^{\infty} x^\sigma \quad = \quad \frac{G(x,\nu-1)}{1-x} \tag{B.10}$$

It is also clear that:

$$G(x,1) \quad = \quad \sum_{\sigma=0}^{\infty} x^\sigma \quad = \quad \frac{1}{1-x} \tag{B.11}$$

Starting with (B.11), applying (B.10) repeatedly and then applying the binomial expansion theorem yields:

$$G(x,\nu) \quad = \quad \frac{1}{(1-x)^\nu} \quad = \quad \sum_{\sigma=0}^{\infty} \binom{\sigma+\nu-1}{\nu-1} x^\sigma \tag{B.12}$$

Comparing (B.12) with (B.9) we can see that:

$$\beta(\sigma,\nu) \quad = \quad \binom{\sigma+\nu-1}{\nu-1} \tag{B.13}$$

Thus we may use (B.13) to compute the length for encoding $(\kappa|k)$:

$$L(\kappa|k) \quad = \quad \log\beta(k,M-1) \quad = \quad \log\binom{k+M-2}{M-2} \tag{B.14}$$

Returning to our original problem of encoding a string of class labels, we have partitioned that code length into the following components:

$$L(s) \;=\; L(k) \;+\; L(\kappa|k) \;+\; L(s|\kappa) \tag{B.15}$$

These terms have the following lengths:

$$L(k) \;=\; log(k_{\max}+1) \tag{B.16}$$

$$L(\kappa|k) \;=\; \log\binom{k+M-2}{M-2} \tag{B.17}$$

$$L(s|\kappa) \;=\; \log\left[\binom{N}{k}\binom{k}{k_1,k_2,\ldots,k_{M-1}}\right] \tag{B.18}$$

$$L(s|\kappa) \;=\; \log\left[\binom{N}{k_1,k_2,\ldots,k_{M-1},(N-k)}\right] \tag{B.19}$$

B.3. Applying Stirling's approximation

Stirling's approximation holds for the factorial function for large values of the argument. It is given by[21]:

$$n! \to \sqrt{2\pi n}\;\left(\frac{n}{e}\right)^n \tag{B.20}$$

Which gives the following for the natural logarithm:

$$\ln n! \;\to\; \frac{1}{2}\ln 2\pi \;+\left(n+\frac{1}{2}\right)\ln n \;-\; n \;\to\; n\ln n \;-\; n \;\to\; n\ln n \tag{B.21}$$

where the forms from left to right are for increasingly larger values of n. The left-most of these forms for $\ln n$ works remarkably well, even for small n as the following table demonstrates. Table 3 compares Stirling's approximation with the "exact" calculation of $\log_2 n$.

C. Enumerating 2^n-Trees

We are interested in enumerating trees for the purpose of encoding them. Specifically we are interested in evaluating our expression for $L(\mu)$ (equation (3.23)), which we repeat here:

$$L(\mu) \;=\; \log\left[\nu_{\max}(2^n,m-1)\;\beta(n_d;2^n,m-1)\;\binom{n_l+M-1}{M-1}\binom{n_l}{q_1,q_2,\ldots,q_M}\right[\tag{C.1}$$

The first term is simply the number of leaves in the largest possible (i.e. saturated) 2^n-tree of maximum depth m, which is given by:

$$\nu_{\max}(a,h) \;=\; \sum_{l=0}^{h} a^l \;=\; \left(\frac{a^{h+1}-1}{a-1}\right) \tag{C.2}$$

The product comprising the last two terms in the argument of the log in (C.1) is derived from the string enumeration formula derived in Appendix B and corresponds to an encoding of the labels of the tree's leaves. What we seek here is an expression for the number of ordered, 2^n-ary trees of maximum depth $m-1$: $\beta(n_d;2^n,m-1)$. This we might refer to as the *tree structure term.*

Table 3: Comparison of Stirling's approximation with exact calculation of $\log n!$. This approximation can be applied to all the enumeration formulas presented here, resulting in expressions that are much simpler to compute.

n	$\log n!$	Stirling's	difference
1	0	-0.1169469762	-0.1169469762
2	1	0.940357983	-0.05964201704
3	2.584962501	2.545031695	-0.03993080613
4	4.584962501	4.554967901	-0.02999459954
5	6.906890596	6.882877382	-0.02401321344
6	9.491853096	9.471834074	-0.02001902224
7	12.29920802	12.28204469	-0.01716332444
8	15.29920802	15.28418774	-0.01502028076
9	18.46913302	18.45578021	-0.01335280939
10	21.79106111	21.77904265	-0.01201846255
11	25.25049273	25.23956623	-0.01092650407
12	28.83545523	28.82543883	-0.01001640099
13	32.53589495	32.52664873	-0.009246224132
14	36.34324987	36.33466386	-0.008586012148
15	40.25014047	40.24212668	-0.008013786549
16	44.25014047	44.24262741	-0.00751305937
17	48.33760331	48.33053209	-0.007071219628
18	52.50752831	52.50084986	-0.006678457158
19	56.75545583	56.7491288	-0.006327026023
20	61.07738392	61.07137319	-0.006010728758

We will change notation to make things slightly less cumbersome and attempt to derive an expression for the number of ordered, a-ary trees with ν nodes and maximum depth h: $\beta(\nu; a, h)$. By *ordered* we mean trees where the position of an edge/child is significant. Thus while there is only one two-node unordered tree, there are two ordered two-node binary trees and a ordered two-node a-ary trees, etc.

Some thought shows that this enumeration function obeys the following recursion relation:

$$\beta(\nu; a, h) \;=\; \sum_{\Lambda} \prod_{i=1}^{a} \beta(\lambda_i; a, h-1) \quad ; \quad \beta(\nu; a, 0) \;=\; 1 \tag{C.3}$$

where

$$\sum_{\Lambda} \;\equiv\; \sum_{\lambda_1=l_1}^{u_1} \sum_{\lambda_2=l_2}^{u_2} \cdots \sum_{\lambda_a=l_a}^{u_a} \tag{C.4}$$

where

$$u_i \;\equiv\; \min\left(\nu_{\max}(a, h-1),\; \nu - 1 - \sum_{j=1}^{i-1} \lambda_j\right) \tag{C.5}$$

and

$$l_i \equiv \max\left(0,\ \nu - 1 - \left[\sum_{j=1}^{i-1} \lambda_j + (a-i)\nu_{\max}(a,h)\right]\right) \quad \text{(C.6)}$$

The recursion relation (C.3) and the other equations come from the following reasoning. Let $\Lambda = \{\lambda_i\}$ be a vector of a components that represent the number of nodes in each of the subtrees attached to the ordered edges originating at the root node for a given tree. The number of a-ary trees of depth h and a fixed Λ is given by: [15]

$$\beta(\Lambda; a, h) = \prod_{i=1}^{a} \beta(\lambda_i; a, h-1) \quad \text{(C.7)}$$

To compute the total number of trees with a given number of nodes, we simply sum this expression over all Λ consistent with ν. That is:

$$\beta(\nu; a, h) = \sum_{\forall_\Lambda > |||\Lambda|| = \nu - 1} \beta(\Lambda; a, h) \quad \text{(C.8)}$$

where:

$$||\Lambda|| \equiv \sum_i \lambda_i \quad \text{(C.9)}$$

The limits on the sums over the $\{\lambda_i\}$, $\{l_i, u_i\}$, are set to ensure that $||\Lambda|| = \nu - 1$. Following the procedure and reasoning used in Appendix B, we will define a generating function $G(x; a, h)$, where

$$G(x; a, h) \equiv \sum_{\sigma=0}^{\infty} \beta(\sigma; a, h)\, x^\sigma \quad \text{(C.10)}$$

Substituting (C.7) into this expression, regrouping and collecting terms give the following recurence relation for generating functions:

$$G(x; a, h) = 1 + x[G(x; a, h-1)]^a \quad \text{(C.11)}$$

There is apparently no known closed-form solution for β or G[19, 20]. Three alternatives for dealing with this situation are:

1. Use (C.3) to compute β recursively.

2. Asymptotic solutions of (C.11) have been studied for the case of $a = 2$ and it appears that those results can be extended to the case of general a [22, 19].

3. A closed-form solution does exist for trees of unlimited depth ($h \to \infty$) and this may be used in a sub-optimal[16] scheme, which may be shown experimentally to be quite adequate.

We will now discuss two of these alternatives.

[15] Please excuse the bastardization of the notation for β.

[16] Using the term "sub-optimal" here implies that using the exact solution for $\beta(\nu; a, h)$ is optimal, which is not clear, strictly speaking. It should be remembered that there is a "leap of faith" involved here. That is the belief that minimizing the description length in the way we are is essentially equivalent to minimizing the error rate on new (not from the training set) samples. This is not clear, and the support for the importance of the technique comes from empirical verification. Rigid belief that using $\beta(\nu; a, h)$ will produce optimal trees leads to the conclusion that using $\beta(\nu; a, \infty)$ will "over penalize" the trees, resulting in over-pruned trees.

C.1. Enumerating Trees of Unlimited Depth

Considering trees of unlimited depth is equivalent to letting $h \to \infty$. In this case equation (C.11) becomes:

$$G(x; a, \infty) \;=\; 1 \;+\; x[G(x; a, \infty)]^a \tag{C.12}$$

The solution to this equation for G can be written: [17]

$$G \;=\; 1 \;+\; x \sum_{\sigma \geq 0} \binom{a(\sigma+1)}{\sigma} \frac{x^{\sigma+1}}{\sigma+1} \tag{C.13}$$

Which, after applying suitable binomial coefficient identities can be reduced to:

$$G \;=\; \sum_{\sigma \geq 0} \left[\frac{1}{(a-1)\sigma+1} \binom{a\sigma}{\sigma} \right] x^\sigma \tag{C.14}$$

From which we conclude that:

$$\beta(\nu; a, \infty) \;=\; \frac{1}{(a-1)\nu+1} \binom{a\nu}{\nu} \tag{C.15}$$

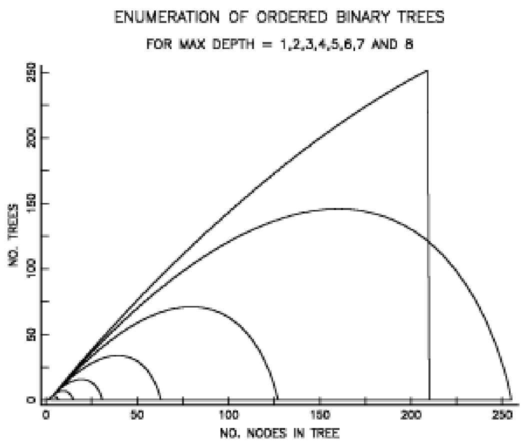
For the case of $a = 2$, β is given by the *Catalan* number[21]:

$$C_\nu \;\equiv\; \frac{1}{\nu+1} \binom{2\nu}{\nu} \tag{C.16}$$

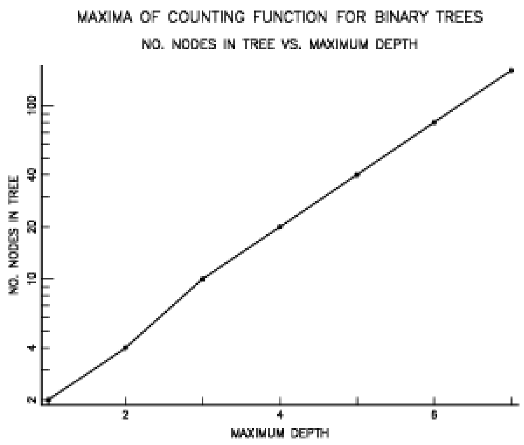
C.2. Direct Recursive Calculation of β

Equation (C.3) can, in principle, be used to compute β for any ν, though this becomes impractical for large ν. A dynamic-programming algorithm was used to compute β for binary trees of various depths. The results of these calculations are shown in Figures 4 and 5.

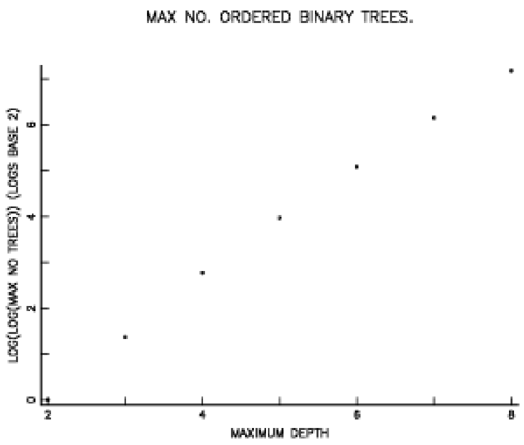
[17] See Knuth[23] Exercise 11 (p.396).



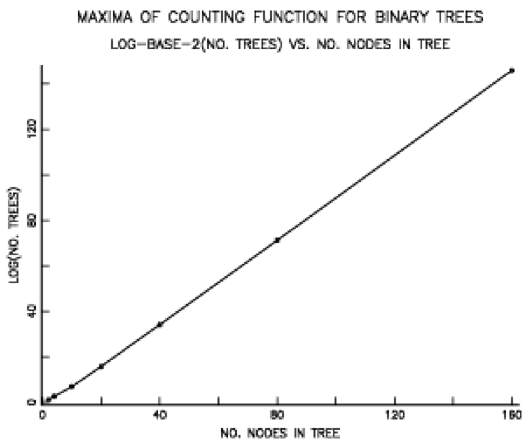
(a)



(b)



(c)



(d)

Figure 4: Enumeration of binary trees. (a) Counting functions for binary trees of various maximum depths; (b) Location of maxima for counting functions for binary trees. (i.e. tree sizes for which β is maximum.); (c) Value of β (number of trees) at maxima in β for binary trees; (d) Maximum value of β (number of trees) vs number of nodes (tree size) for binary trees.

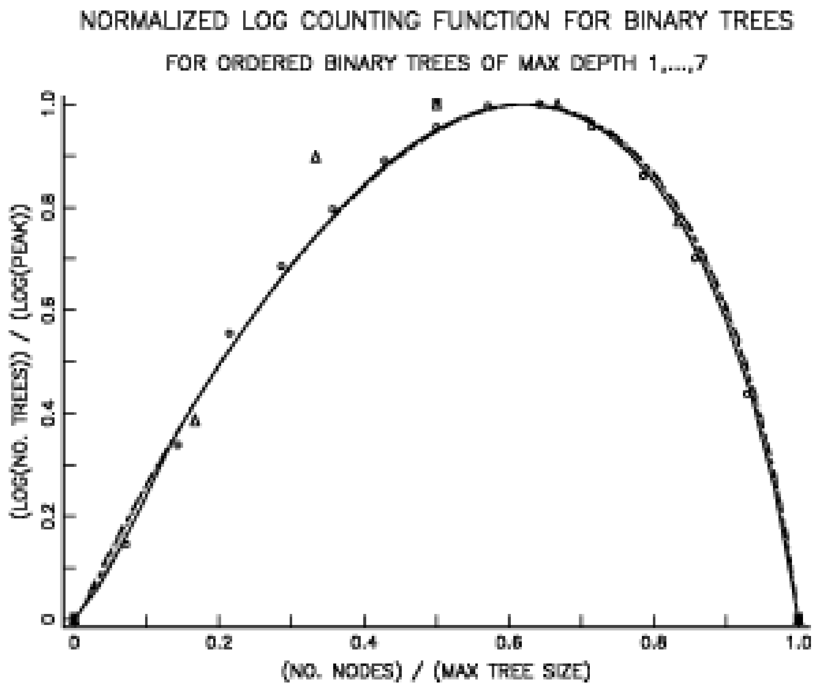


Figure 5: Normalized counting functions for binary trees of various maximum depths.