Realtime Image Segmentation with 2^n-Tree Classifiers

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

Abstract

For realtime classification applications (e.g. realtime image segmentation), the number of usable pattern classification algorithms is limited by the feasibility of high-speed hardware implementation. This paper describes a non-parametric pattern classifier and associated hardware architecture and training algorithms. The classifier has both a feasible hardware implementation and other desirable properties not normally found in statistical classifiers. In addition to the classification/training algorithms and hardware architecture, the paper discusses the application of the technique to the problem of image segmentation. Results from segmenting images are included.

The scheme described has two major aspects: (1) The classifier itself, which is implemented as a 2^n-tree, a hierarchical data structure that corresponds to a recursive decomposition of feature space and (2) Training schemes, specific to the 2^n structure, by which the classification tree is constructed. The training algorithms discussed have the following important properties:

1. *They are non-parametric and therefore independent of any particular probability model (e.g. Gaussian).*
2. *They can handle any shaped decision regions in feature space.*
3. *They are consistent in the sense that for large training data sets they produce a classifier that approaches the ideal Bayes classifier.*

The training algorithms also include an interesting application of the Minimum Description Length principle (MDL). It is used in a tree pruning algorithm that produces trees that are both significantly smaller and, at the same time, have better classification performance (i.e. lower error rates) than unpruned trees.

The architecture affords a practical hardware implementation that can also be used to implement other classification schemes.

1 Introduction

This paper describes a scheme for doing machine classification and learning and its application to realtime image segmentation. A more detailed treatment of this work can be found in [4]. An investigation of appropriate hardware implementations is done in [14].

In what follows we assume that objects (e.g. patterns) are being classified by performing a set of *feature* measurements thus forming a *feature vector* ($x = \{x_i\}$) corresponding to the object. These feature vectors identify points in *feature space*. In the training procedure the feature space is segmented into various regions corresponding to the various classes of objects. The task of the classifier is to produce a label for feature vectors generated from future instances of objects not included in the training. In our segmentation application the classifier outputs images with pixel values corresponding to the class label. With a hardware implementation these labeled images would be produced at video rates.

1.1 Previous Related Work

While other tree-based classifiers have been proposed and studied[6, 2, 8, 7], it appears that the 2^n tree has never been proposed as an architecture for a pattern classifier. Omhohundro [7] does suggest it as a data structure for storing training data to be used by algorithms such as "k nearest neighbor"[3]. The 2^n tree has been used as a data structure for other problems, however. The most common case being $n = 2$, commonly referred to as a *quadtree*, which has been used in many image processing and analysis applications as a data structure for representing an image. Samet[13] has reviewed such quadtree applications extensively. His review also mentions some applications for $n > 2$. None of these resemble those proposed here, however. Approaches to using criteria based on the

Samet[13] has reviewed such quadtree applications extensively. His review also mentions some applications for $n > 2$. None of these resemble those proposed here, however. Approaches to using criteria based on the Minimum Description Length (MDL) principle[11, 10] to prune general decision trees have been described in [9, 12, 11]. We have adapted and extended these approaches to cover the case of 2^n trees. A comparison of the 2^n tree and the associated training algorithms with those presented by Brieman et. al.[6] appears at the end of this paper.

2 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 level $m-1$ nodes, all m-th level nodes being leaves. For $n = 2$ the tree takes the form of the well-known quadtree data structure.

At classification time the tree acts as a kind of look-up table (LUT) in the sense that, when 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 Figure 1. The case shown here is for two features and two classes (i.e. $n = 2$ and $M = 2$).

3 Tree Construction Algorithms

We give a brief description of some different training algorithms we have investigated. The SLP and Monte Carlo schemes rely on a previously known classifier that will be mapped to the 2^n-tree architecture. The GLF and SS algorithms are non-parametric techniques that "learn" the tree structure from training data. For more details see [4].

3.1 Straightforward LUT Programming (SLP)

In this case it is assumed that a classifier has been developed and exists in some form that produces desired (or at least acceptable) class assignments, but it cannot be used in practical applications due to any or all of cost, execution time or memory requirement. A 2^n tree can be constructed that implements the known classifier by an algorithm (described in [4]) that visits every *atomic* cell in feature space. While this algorithm can always be used in principle, it becomes impractical for high-dimensional feature spaces because of the time required to generate the tree.

3.2 Monte Carlo Schemes

In cases where one desires to program the tree with a definite parametric classifier, but the size of the feature space prohibits use of the SLP algorithm, a Monte Carlo approach may be used. For instance a Gaussian model could be made and used to generate synthetic feature vectors that can be input to one of the learning algorithms below.

3.3 Grow Leaves First (GLF)

This algorithm consists of a single growing phase followed by a pruning phase. In the growing phase the complete path to the bottom-level leaf corresponding to each training vector is created. Class histograms are accumulated for the bottom-level leaves during this process. The tree is then pruned recursively with leaves whose siblings all have the same class as theirs being pruned, creating a new leaf (i.e. the former parent). The new leaf's class histogram is the sum of all the previous children's histograms.

3.4 Successive Specification (SS)

In cases where the storage required for intermediate trees by the GLF algorithm is prohibitive the SS algorithm may be used. In this algorithm the classification tree is constructed in a manner that resembles the human learning process. Initial information is generalized to cover a large number of cases and is then modified making it more specific to reflect subsequent information. This process proceeds as follows. The

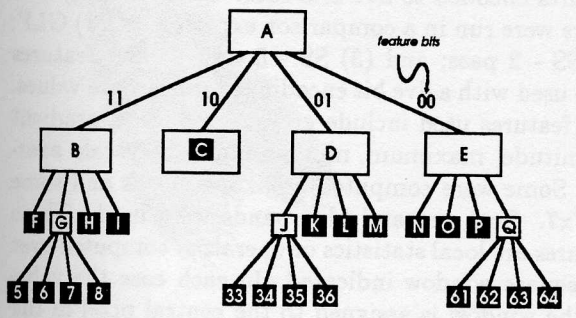

Figure 1: a: Example of a small 2^2 tree of depth 3. Decision nodes are in white; leaf nodes in black.

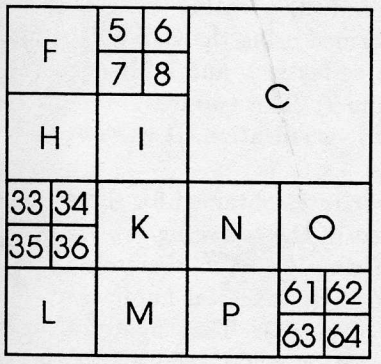

Figure 1: b: Subdivision of the 2-dimensional feature space by this tree.

first training vector is used to set the class of the root node so that, if the tree were to be used to perform classification at that point, all vectors would be classified as that same class. Subsequently, for each training vector, the tree is searched to find the leaf corresponding to x. If the class of that leaf is the same as that of the vector being processed, nothing happens. If the class is different, on the other hand, the children of the node are added to the tree, all retaining the class of the parent except the one corresponding to x, which is given the class corresponding to x.

This process of growing new branches and adding leaves to accommodate new training data is adequate for separated classes (i.e. no overlap in feature space). If there is overlap, however, two or more vectors with the same x but different classes may appear in the same training set. To accommodate such cases this algorithm creates (and subsequently maintains) a class histogram for each bottom-level leaf visited more that once during the training process. The class receiving the most counts is assigned to the leaf at the completion of the training process.

Improved performance at the expense of additional memory and execution time in the training phase, can be obtained by maintaining class histograms at all nodes (not just bottom-level leaves). The class associated with a given node is then taken to be the class whose histogram bin is the largest. If the node is not a leaf, then the class associated with children that have never been "visited" is determined by the class of the node (i.e. the parent of those children). This technique will be used in the algorithm and associated results presented here.

A problem with the SS procedure as thus far described is that the effect of training vectors on the ultimate classification tree is strongly dependent on the order in which they are processed. To correct this situation multiple passes through the training data are made. To ensure that all non-ambiguous [2] training vectors will be classified correctly the number of passes through the training data that must be made is one plus half the tree depth (which is equal to the number of bits of precision being used for the feature values)[4]. In practice, however, two or three passes appear adequate.

3.5 SS/GLF Hybrid Algorithm

The classifier produced by the GLF algorithm has desirable theoretical properties and is in fact the classifier that the SS algorithm, in a sense, attempts to

[2] *Ambiguous* training vectors are those for which the training set contains at least two identical vectors with different class assignments.

approximate. On the other hand, the GLF algorithm is more susceptible to memory limitation problems. In [4] we describe an an SS-like algorithm that will produce the GLF classifier in a relatively small number of passes through the training data.

4 Pruning the tree

In general the training processes described above will produce trees where, in many cases, all the children of a given node will be leaves with the same class. In this case storage requirement and run time are improved by recursively pruning such unnecessary leaves as described above. As we describe below, a further pruning of the tree using an MDL-based technique produces significantly smaller trees that actually perform classification better than their unpruned counterparts.

4.1 MDL-Based Pruning

Due to the high number of degrees of freedom inherent in the 2^n-tree classifier, a large number of training vectors would be required to produce a classifier that did not suffer from over-fitting the training data. To significantly reduce this requirement we have developed an information-theoretic pruning algorithm. This technique is an extension of that proposed by Quinlan and Rivest[9]. It is based on the *Minimum Description Length principle* (MDL), proposed by Rissanen[10] 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 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. A detailed description of this technique may be found in [5].

5 Classification Results

The SS and GLF algorithms and the resulting classifiers were used to segment [3] multi-band images. Two spectral bands were used - red and blue - and eight features encoded to five bits each. The following classifiers were run in a comparison experiment: (1) GLF; (2) SS - 2 pass; and (3) SS - 5 pass. Eight features were used with a five bit encoding of the feature values. The features used include greyscale average, gradient magnitude, maximum, minimum and greyscale average. Some were computed on 5x5 windows and some on 7x7. Both red and blue bands were used. These features are local statistics or operators computed over the square window indicated. In each case the value for the window is assigned to the central pixel in the feature image. The precise definitions of the features are unimportant for the purpose of this exposition and will be omitted.

The classifiers were trained on one image. Then two images were segmented: the training image and another image acquired under the same conditions as the training image. To help visualize the nature of the classifier formed by the training algorithms two figures are included. They are (1) a 2-D scatter plot formed using the training mask and the two bands (red and blue) as features (Figure 2) with 8 bits of resolution and (2) a feature space diagram of the tree classifier formed using the GLF algorithm, only these two bands as features and 5 bit encoding of the features (Figure 2). This two-feature tree is included only as an aid to visualization. It was not used to segment the images.

The error rates obtained for these experiments are summarized in the following table. The "total error rates" are simply the total number of misclassifications divided by the total number of feature vectors (i.e. no. of pixels). The "figure of merit" numbers are computed by equal weighting of the various error rates and by ignoring confusion between the edge class and adjacent classes (in the training mask) in a nar-

[3] By *segmentation*, in this context, we mean the process of classifying the pixels in a digital image. There is some small number of classes of interest (~ 4) specified by training masks (hand segmented images) and associated images. Image data for more than one band is provided. In the examples provided two spectral bands are used (red and blue). The classification is based on features that are statistics computed over some local neighborhood about the pixel to be classified. For example, one feature might be the average graylevel of the red band in a 7x7 pixel square window.

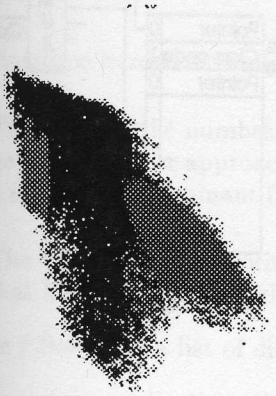

Figure 2: a: Scatter plot of the red and blue bands of the image used.

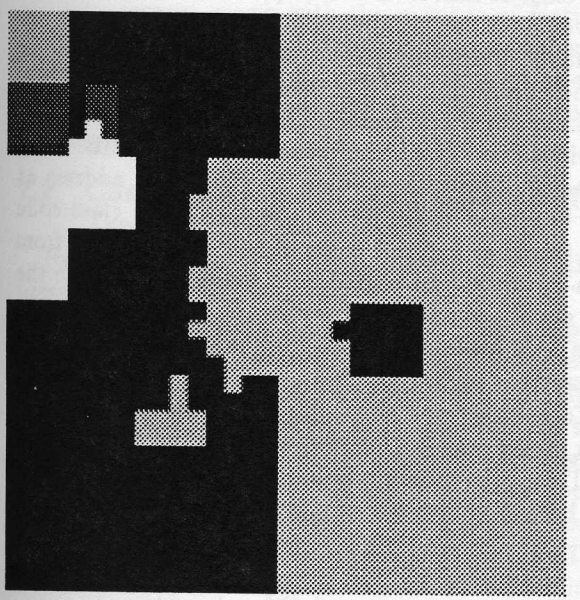

Figure 2: b: A feature-space plot of the decision regions corresponding to the 2^n-tree classifier produced using the GLF algorithm on the red and blue bands (i.e. just two features) coded to five bits.

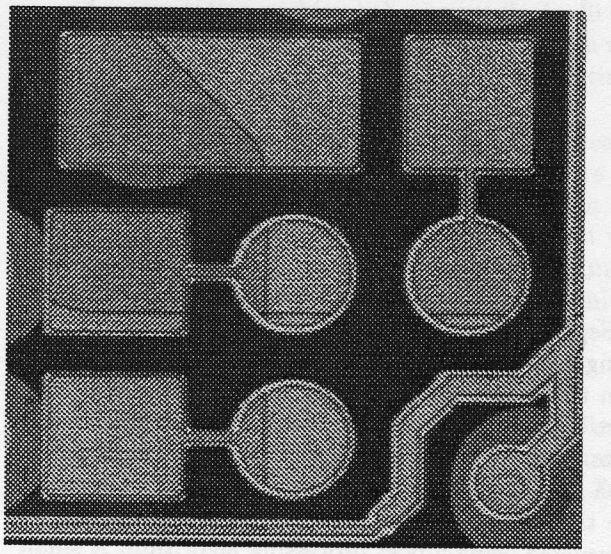

Figure 3: a: The red band of a typical image from our segmentation application.

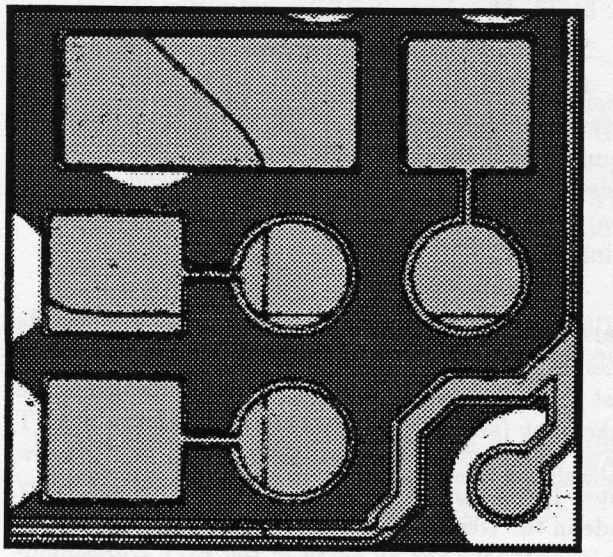

Figure 3: b: The above image segmented using a 2^n-tree classifier and the eight features mentioned in the text.

row zone around region boundaries. This number is intended to be more relevant to the particular segmentation problem than total error rate. It also mitigates the confusion inherent between edge class[1] and other classes.

Tree Classifier Segmentation Results		
GLF	SS	SS
passes 1	passes 2	passes 5
depth 5	depth 5	depth 5
features 8	features 8	features 8
3.71 % error	3.79 % error	3.77 % error

Segmentation of these images is difficult, especially using only local information. There is inherent confusion that can't be eliminated by any classifier. They were chosen to demonstrate the ability of the tree classifier as trained by the SS and GLF algorithms to deal with complex and difficult classification problems. In the target application the segmentation task will be completed using a post processing algorithm. A small training set (one image) and only five bit feature encoding were used to save computation time during training. An extensive experiment is planned to determine typical tree sizes (as information for potential hardware implementation) and to compare these classifiers with others.

6 Software and Hardware Implementations

Software: The algorithms described have been implemented in software, using the C language running under AIX on an IBM *RISC System 6000* workstation. All algorithms implement the tree structure as a linked list.

Hardware: We have investigated several possible schemes for a hardware implementation [14] and the best candidate was found to be kind of hardware linked list (see Figure 4). This figure shows a LUT RAM for every level in the tree (though the first few upper levels can be combined). In this scheme every node in the tree (even those whose class is determined by inheritance) has an entry in the LUT corresponding to its level. The entry contains a flag indicating leaf or decision node, the class code (for leaves) or a pointer to the first child (for decision nodes) in the next level LUT. These last two items can obviously occupy the same space.

[1] An *edge* class was defined corresponding to the boundary regions between the other class regions. This is discusses in more detail in [1].

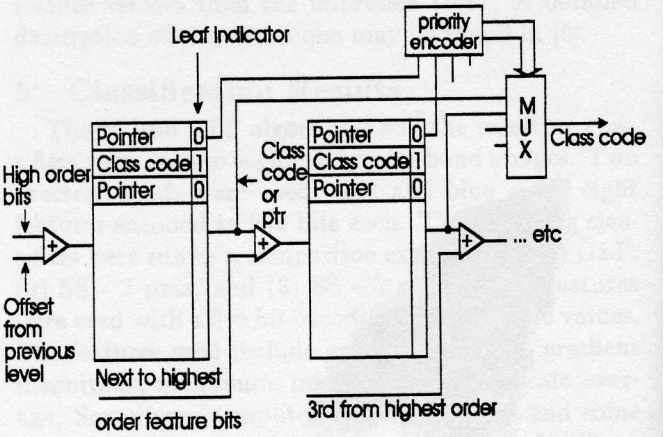

Figure 4: Hardware-linked-list implementation of a 2^n-tree classifier.

The first LUT corresponds to level one and contains an entry for every node at that level (one node trees are precluded). For levels below that, only the nodes that exist at that level will be present. The children of a given node are grouped together and the groups are listed one after the other corresponding to the ordering of the parent nodes. The adder at the input to each level combines the address of the first child from the previous level (i.e. the parent) with the bits being tested at this level to determine the node address at this level and so on, down the tree. The class code bits (same as at least some of the address bits) from each level are also routed to a multiplexor where the leaf status bits from all levels are used to select the class code from the highest level node for which "leaf" is asserted.

To achieve high throughput, results from the various stages will be latched and clocked through the LUT array in pipeline fashion. Such details are omitted here, but they are discussed in [14].

7 Discussion

7.1 Advantages and Disadvantages

The following is a list of the advantages of this approach over many others. Both this list and the list of disadvantages below it focus on the SS and GLF training algorithms. Though some comments obviously refer to the tree architecture in general.

2. No prior knowledge about the shape of the distributions is required. This coupled with (1) makes 2^n trees an excellent approach in applications where an automatically trainable classifier is required.

3. Realtime hardware implementation is feasible.

4. The effect of the number of classes is minor compared with other approaches where each class has a separate discriminant function.

5. The GLF and SS algorithms produce classifiers that become the Bayes classifier as $N \rightarrow \infty$.

The following is a list of disadvantages.

1. In certain applications this approach will require much more memory for the classification and/or training process than some other approaches. For example, if a Gaussian classifier performs adequately in a certain application, it will certainly take significantly less storage to implement it as compared with the associated 2^n tree. On the other hand, the tree may execute faster during classification in software implementations.

2. A problem for the 2^n tree is the way its storage requirement grows with the number of features. This makes it useful only for small numbers of features ($n \leq 10$). Once again, this is quite acceptable for image segmentation and many other applications.

7.2 Comparison with the Classification Trees of Brieman et.al.

Here we compare the 2^n-tree algorithms with those presented in [6]. We will refer to these as *CART* (for *Classification and Regression Trees*). For the purposes of comparison with our approach CART captures all the important aspects of previous tree-based classifiers. In growing 2^n trees the only local (on the tree) decision to be made is one of whether or not to *split* a certain node, thus creating its children. If the node is split, the test to be performed there [6] is determined by the position of the node within the tree. In CART, on the other hand, one of the central issues is determining the test to be performed at a given node.

The complexity of CART is considerably more than GLF training a 2^n tree, when considered as a function of the number of training vectors. On the other hand, one might expect CART to obtain better (in terms of classification error) trees for a given number of training vectors. In cases where large number of training vectors are available at relatively low cost (e.g. our image segmentation problem) these two effect will offset each other and it is not clear which algorithm will produce the better trees for a constant training time. Both algorithms are consistent.

For hardware implementation the 2^n tree is clearly preferable. No comparisons or multiplications are required, only testing of certain feature bits, which act as address lines to RAM LUTs (in the linked-list implementation). Also, 2^n trees are of fixed maximum depth, while CART trees can be of virtually unlimited depth.

[6] In the case of 2^n trees the test consists of testing certain feature bits and the bits to be tested are determined by the level of the node within the tree.

References

[1] B.Dom, G.Healey, D.Steele, A.Dorundo W.Blanz, and D.Capson. Algorithms for realtime image segmentation. In *Proceedings of VISION 90*, July 1990. Also IBM Research Report RJ7581.

[2] P. Chou. Applications of information theory to pattern recognition and the design of decision trees and trellises. *Ph.D. Dissertation, Stanford Univ.*, 1988.

[3] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Trans Inf Th*, IT-13:21–27, 1967.

[4] Byron Dom and David Steele. 2^n−tree classifiers and realtime image segmentation. RJ 7558, IBM Research Division, 1990. A shorter version also appears in the proceedings of the 1990 IAPR Workshop on Machine Vision and Applications; Tokyo, Japan.

[5] Byron Dom and David Steele. MDL-based pruning of 2^n−tree classifiers. RJ 8673, IBM Research Division, 1992.

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

[7] S.M. Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1:273–347, 1987. Also Univ. Illinois CS Report no. UIUCDCS-R-87-1331 (4/87).

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

[9] J.R. Quinlan and R. L Rivest. Inferring decision trees using the minimum description length principal. *Information and Computation*, 80:227–248, 1989.

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

[11] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15. World Scientific Series in Computer Science, 1989.

[12] J. Rissanen and M. Wax. Algorithm for constructing tree structured classifiers. 1988. U.S. Patent No. 4,719,571, 1/12/88.

[13] Hanan Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2), 1984.

[14] Patrick Wambacq, Byron Dom, and David Steele. Realtime hardware architecture for 2^n−tree classifiers. RJ 8457, IBM Research Division, 1991. A shorter version of this appears in the proceedings of CAMP91.