*MVA'90* IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

# $2^n$-Tree Classifiers and Realtime Image Segmentation

**Byron E. Dom and David Steele**

**IBM Research Division**
**Almaden Research Center**
**650 Harry Road**
**San Jose, California 95120-6099**

## ABSTRACT

For realtime pattern 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 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 a look-up-table (LUT) implemented as a $2^n$-*tree*, which is 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. These training schemes may be used as techniques for machine learning. Two of the training algorithms have the following important properties: they are non-parametric and therefore independent of any particular probability model (e.g. Gaussian); they can handle any shaped decision regions in feature space; and They are *consistent* in the sense that for large training data sets they produce a classifier that approaches the ideal Bayes classifier. These attributes make this architecture/algorithm combination an excellent alternative to artificial neural networks, a class of classifiers in which there has been much interest, of late. 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.

## INTRODUCTION

The following are what we consider to be the contributions of the work described in this paper. A more detailed treatment of this work can be found in [1].

1. The use of the $2^n$-tree as a model/architecture for classification.
2. Four classifier training algorithms. Only one will be described here.
3. Experimental verification of the effectiveness of these algorithms.
4. An outline of possible hardware implementations of the $2^n$-tree.
5. A proof that the GLF (and certain others) algorithm produces classifiers that become the Bayes classifier for large ($N \rightarrow \infty$) training sets.

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*. The feature space is either explicitly or implicitly segmented into various regions corresponding to the various classes of objects. The task of the classifier, in this model, is to identify the region containing the feature vector and thereby label it (the vector), while the task of the training procedure is to perform the segmentation of feature space and represent it in a form usable by the classifier. To simplify the language below we will refer to the process of classifying feature vectors.

### PREVIOUS RELATED WORK

While other tree-based classifiers have been proposed and studied (see [2, 3,4,5]) it appears that the $2^n$-tree has never been proposed as an architecture for a pattern classifier. Omhohundro[5] does suggest it as a data structure for storing training data to be used by algorithms such as *K nearest neighbor*[6]. 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[7] 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, 12] to prune general decision trees have been described in [8,9, 10,11]. We have adapted and extended these approaches to cover the case of $2^n$-trees.

## 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: $\{x_1, x_2, ...\}$, 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.

## TRAINING ALGORITHMS

**Straightforward LUT programming (SLP):** In this case any classifier training scheme can be used. 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. In this case, this classifier will be used to program the $2^n$-tree, which will then be used to perform the actual classification. Denote this classifier by $\gamma(x)$. The $2^n$-tree is programmed using $\gamma(x)$ as follows. In this procedure every cell in the discrete feature space is visited. These cells are processed in an order equivalent to representing the entire feature space with a complete $2^n$-tree. In this tree all leaves ($2^{mn}$ of them) will be $m$-th level nodes and all $m$-th level nodes will be leaves. The total tree consists of $(2^{n(m+1)} - 1)/(2^n - 1)$ nodes. This abstract tree is then processed recursively in *post order*. 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.

**Monte Carlo Schemes:** In cases where one desires to program the tree with a definite classifier, but the size of the feature space prohibits use of the SLP algorithm, a Monte Carlo approach may be used, where synthetic training vectors are generated using the desired classifier. The training vectors so-produced are then used as input to a direct or "learning" algorithm (described below).

### TREE "LEARNING" ALGORITHMS: GROW LEAVES FIRST (GLF)

Only the conceptually simplest training algorithm will be described here. Others are described in [1]. 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.

After formation of the initial tree, but before pruning, the tree contains the *joint* histogram $H(c, x)$ and the class returned by this tree (or its pruned counterpart) for a given feature vector $x$ during classification will be the class for which $H$ is maximum, if $x$ was among the training vectors. If $x$ was not among the training vectors, its class is determined by association (via the pruned tree) with other vectors within the quad corresponding to the leaf of the pruned tree that corresponds to $x$. This classification has a "K nearest neighbor" flavor, but the neighbors within some quad containing the feature vector are used rather than the K nearest, via Euclidean distance, grid distance or some other metric. Another way to view this is to say that $H(c, x)$ is always used, but that the features are measured to different numbers of bits of precision in different regions of feature space.

A practical problem in using the GLF algorithm is that it may (for high dimensional feature spaces and certain class structures) produce very large unpruned trees. A technique to significantly reduce this problem is to sort the training data, prior to training, into a sequence corresponding to a post-order traversal of the tree corresponding to the entire feature space. This will allow pruning to take place in an on-going fashion. Other training algorithms that

allow memory requirement to be traded against execution time are described in [1].

**Pruning the tree:** After the class histograms are used to set the appropriate node classes, the tree is pruned. This is done by processing the tree in postorder. In this process if all the children of a node are leaves of the same class, those children are eliminated from the tree and the node is converted to a leaf of the appropriate class. This is carried out recursively so that a node with several layers of structure under it may, in principle, be eliminated.

**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[8]. It is based on the *Minimum Description Length principle* (MDL), proposed by Rissanen[12] 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.

**The importance of feature scaling:** Smaller classification trees and better classifier performance will be obtained by scaling the feature values to fill the available dynamic range. For example, if the values of some feature ranging between 0 and 32 are encoded in 8 bit bytes, the high order 5 bits should be used rather than the low order 5. A longer discussion of these effects appears in [1].

## CLASSIFICATION RESULTS

The GLF algorithm and the resulting classifiers were used to segment multi-band images. Two spectral bands were used - red and blue - and eight features encoded to five bits each. These features are local statistics or operators computed over the square windows up to 7x7 pixels in size. These operators include things like local min, max and mean (see [1] for details.

In each of several cases the classifiers were trained on one image; then two images were segmented: the training image and another image acquired under the same conditions. Quite good segmentation results were obtained. Error statistics and sample images and segmentation results are presented in [1].

## SOFTWARE AND HARDWARE IMPLEMENTATIONS

**Software:** The algorithms described have been implemented in software, using the C language running under AIX on an IBM RS-6000 workstation with 48Mbytes of memory. All algorithms implement the tree structure as a linked list. The structure of the tree is the same for both training and classification, but much more information is stored at each node during training. In both cases each node contains a flag indicating whether the node is a leaf or a decision node, a class label and decision nodes contain a list of pointers to their children. During training, a class histogram is also stored at each node.

**Hardware:** Two possible hardware implementations are proposed here. The first will be referred to as D/E for *"decoder/encoder"*. In this scheme each leaf in the tree has a column of AND gates in the decoder array. The idea here is that each leaf has a unique signature and the encoding of the upper leaves (fewer bits) can never be a prefix of the encoding of some lower level leaf. All the bits (for 1's) or inverted bits (for 0's) corresponding to a given leaf are ANDed together to produce one of the decoder outputs. Each of these outputs is an input to a very large encoder that produces a code corresponding to a simple enumeration of the leaves, requiring $\log_2$(maximum no. leaves) bits. This encoded leaf index is then the input to the final stage of the classifier, which is a LUT, containing the class assignments of all the leaves. Such a scheme could be implemented using a programmable logic array (PLA), for example.

The second hardware implementation is a kind of hardware linked list where there is 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.

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. An 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, however.

## DISCUSSION

**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 GLF training algorithm. Though some comments obviously refer to the tree architecture in general.

1. The classifier can handle any shape decision surfaces and any feature vector distributions ( $P(x \mid c)$).
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 algorithms (as well as others described in [1]) produces 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 slight shift in the position of a decision boundary may cause a large increase (or decrease) in tree size in certain cases. This is an artifact of the $2^n$-tree structure and should not be a problem as long as the computing resources available are adequate to handle the worst case. Also, such cases are somewhat pathological. For example, such large changes won't occur for complicated decision boundaries.
3. In cases where limited training data is available and there is inherent confusion between classes, the high number of degrees of freedom available in the $2^n$-tree may result in overtraining (i.e. so called "fitting the noise") effects. It seems that it should be possible to develop an information theoretic pruning technique to handle this, however. See [8], for example.

**Feasibility of Hardware Implementation:** For the examples presented in the experimental results section above the tree sizes ($\sim 10^4$ nodes) make hardware implementation clearly feasible. For larger training sets ($\sim 10^6$ vectors) and more bits per feature (say 8) the trees will be larger, but they are bounded in size by a number proportional to the number of training vectors and, as shown in [1], above a certain $N$, the tree structure stops changing. With the state of the art in memory technology it seems likely that an implementation capable of handling many practical applications is feasible.

## REFERENCES:


1. B. Dom and D. Steele, "$2^n$-Tree Classifiers and Realtime Image Segmentation", *IBM Research Report RJ 7558 (70424)*, (July 2, 1990)
2. L. Brieman, J. Friedman, R. Olshen and C. Stone, *Classification and Regression Trees*, Wadsworth International group (Belmont, CA) (1984)
3. P. Chou, "Applications of Information Theory to Pattern Recognition and the Design of Decision Trees and Trellises", *Ph.D. Thesis, Stanford University* (1988)
4. J.R. Quinlan, "Induction of Decision Trees", *Machine Learning*,**1**, (1986) 81-106
5. S.M. Omohundro, "Efficient Algorithms with Neural Network Behavior", *Complex Systems*,**1**, (1987) 273-347
6. T.M. Cover and P.E. Hart, "Nearest Neighbor Pattern Classification", *IEEE Trans Inf Th*, **IT-13:1** (1967) 21-27
7. "The Quadtree and Related Hierarchical Data Structures", Hanan Samet, *Computing Surveys*, **16**,(2),(6/84)
8. J.R. Quinlan and R.L. Rivest, "Inferring Decision Trees Using the Minimum Description Length Principle", *Information and Computation*, **80**, (1989) 227-248
9. J. Rissanen and M. Wax, "Algorithm for Constructing Tree Structured Classifiers", *U.S. Patent No. 4,719,571*, 1/12/88
10. M. Wax, "Construction of Tree Structured Classifiers by the MDL Principle", *Proc ICASP*, (1990) 2157-2160
11. J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, Vol. 15 in the World Scientific Series in Computer Science, ISBN 9971-50-859-1 (1989),
12. J. Rissanen, "Modeling by Shortest Data Description", *Automatica*, **14**, (1978) 465-471