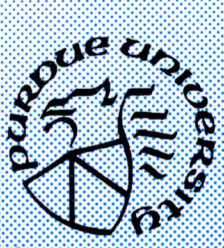
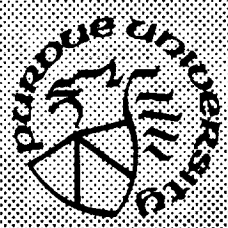


A Survey of Decision Tree Classifier Methodology

S. Rasoul Safavian
David Landgrebe

TR-EE 90-54
September, 1990

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907



A Survey of Decision Tree Classifier Methodology

S. Rasoul Safavian
David Landgrebe

TR-EE 90-54
September, 1990

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TABLE OF CONTENTS

A SURVEY OF DECISION TREE CLASSIFIER METHODOLOGY[1]

by

S. Rasoul Safavian and David Landgrebe
School of Electrical Engineering
Purdue University, West Lafayette, IN 47907

Abstract - Decision Tree Classifiers (DTC's) are used successfully in many diverse areas such as radar signal classification, character recognition, remote sensing, medical diagnosis, expert systems, and speech recognition, to name only a few. Perhaps, the most important feature of DTC's is their capability to break down a complex decision-making process into a collection of simpler decisions, thus providing a solution which is often easier to interpret. This paper presents a survey of current methods for DTC designs and the various existing issues. After considering potential advantages of DTC's over single stage classifiers, subjects of tree structure design, feature selection at each internal node, and decision and search strategies are discussed. Finally, several remarks are made concerning possible future research directions.

I. INTRODUCTION

The decision tree classifier is one of the possible approaches to multistage decision making; table look-up rules [35], decision table conversion to optimal decision trees [37],[44],[58],[94], and sequential approaches [22],[80] are others. The basic idea involved in any multistage approach is to break up a complex decision into a union of several simpler decisions, hoping the final solution obtained this way would resemble the intended desired solution. A complete review of *multistage* recognition schemes is given by Dattatreya and Kanal [14]. Some of the differences between the different multistage schemes are addressed in Kulkarni & Kanal [46]. The emphasis of this paper is on the hierarchical approaches. Hierarchical classifiers are a special type of multistage classifiers which allow rejection of class labels at intermediate stages.

The organization of the paper is as follows: Section II contains the preliminaries, definitions, and terminologies needed for later sections; section III explains the motivations behind DTC's and their potential use and drawbacks; section IV addresses the problems of tree structure design, feature selection and decision rules to be used at each internal node. Tree structure design is considered from top-down, bottom-up, hybrid, and tree growing-pruning points of view. In considering the

1 The work reported in this paper was supported in part by NSF Grant ECS 8507405

feature subset selection at the internal nodes of the tree, a newly proposed approach based on application of neural networks is considered. Some possible search strategies are also mentioned in this section. These problems are addressed from the Bayesian, the decision-theoretic, and the information theoretic and entropy reduction approaches. Other issues such as incremental tree design, the tree capability to generalize, the missing data value problem, the robustness of tree design, and the relation between decision trees and neural networks are discussed in section V. Final comments and conclusions are provided in section VI.

II. PRELIMINARIES

We briefly describe some necessary terminologies for describing trees (see Aho et. al. [3] for more details).

Definitions :

1) A graph G = (V, E) consists of a finite, non-empty set of *nodes (or vertices)* V and a set of *edges* E. If the edges are ordered pairs (v,w) of vertices, then the graph is said to be *directed.*
2) A path in a graph is a sequence of edges of the form (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n) . We say the path is from v_1 to v_n and is of the length n.
3) A directed graph with no cycles is called a *directed acyclic graph.* A *directed* (or *rooted*) *tree* is a directed acyclic graph satisfying the following properties:

   i) There is exactly one node, called the *root*, which no edges enter. The root node contains all the class labels.
   ii) Every node except the root has exactly one entering edge.
   iii) There is a unique path from the root to each node.

4) If (v, w) is an edge in a tree, then v is called the *father* of w, and w is a *son* of v. If there is a path from v to w ($v \neq w$), then v is a proper *ancestor* of w and w is a proper *descendant* of v.
5) A node with no proper descendant is called a *leaf* (or a *terminal*). All other noes (except the root) are called *internal* nodes.
6) The *depth of a node* v in a tree is the length of the path from the root to v. The *height of node* v in a tree is the length of a largest path from v to a leaf. The

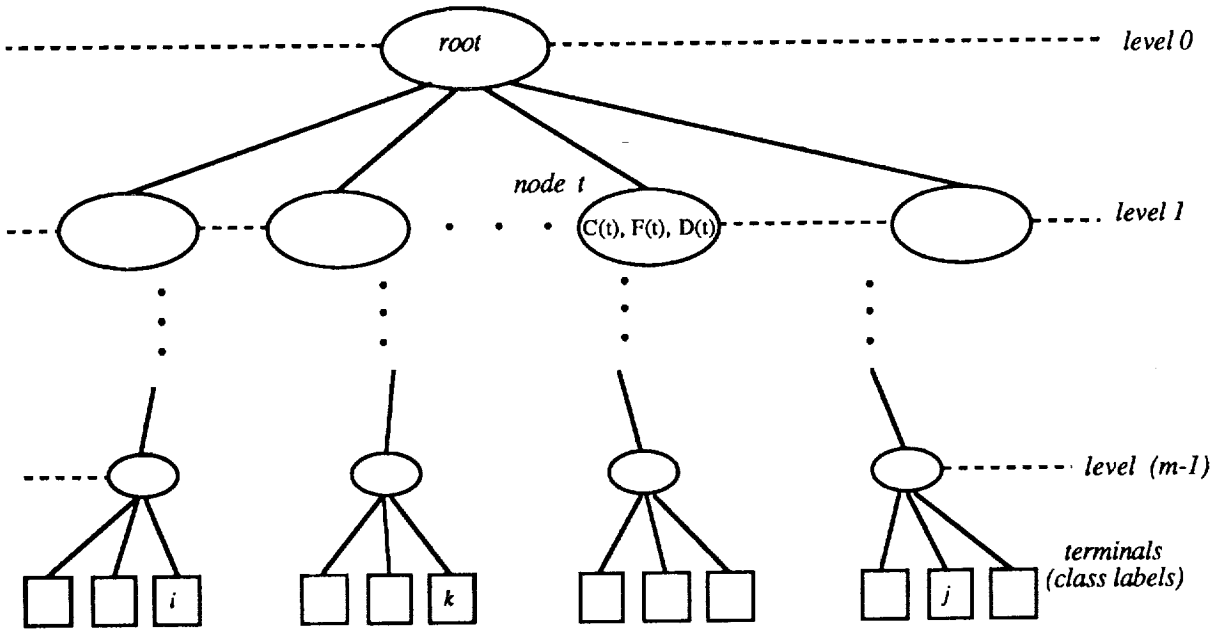
height of a tree is the height of its root. The *level* of a node v in a tree is the height of the tree minus the depth of v.

7) An *ordered tree* is a tree in which the sons of each node are ordered (normally from left to right).

8) A *binary tree* is an ordered tree such that
   i) each son of a node is distinguished either as a *left son* or as a *right son*, and
   ii) no node has more than one left son nor more than one right son.

9) The balance of a node v in a binary tree is $(1+L)/(2+L+R)$, where L and R are the number of nodes in the left and right subtrees of v. A binary tree is α-*balanced* , $0 < \alpha \leq 1$, if every node has balance between α and $1-\alpha$. We call a tree with $\alpha=1$ a fully balanced tree or simply a *balanced* tree. A fully balanced tree is also known as a *complete* tree.

We will denote any tree by T. See Figure 1 for an example of a general complete tree.



C(t) - subset of classes accessible from node t
F(t) - feature subset used at node t
D(t) - decision rule used at node t

Figure 1. Example of a general balanced (complete) decision tree.

10) If two internal nodes contain at least one common class, then it is said that the nodes have *overlap* classes. See Figure 2.

11) The average number of layers (or levels) from the root to the terminal nodes is referred to as the *average depth* of the tree. The average number of non-terminal nodes in each level of the tree is referred to as the *average breadth* of the tree. In general, the average breadth of the tree will reflect the relative weight given to classifier accuracy whereas the average depth of the tree will reflect the weight given to efficiency [78].

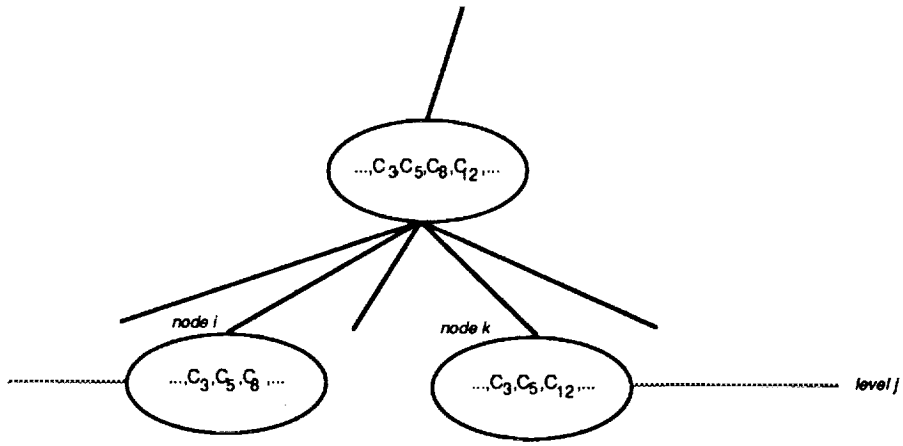


Figure 2. Part of the tree with overlap at nodes i and j; i.e., classes 3 and 5 are repeated.

Let us also introduce the following. Let ($\underline{\mathbf{X}}$, **Y**) be jointly distributed random variables with q-dimensional vector $\underline{\mathbf{X}}$ denoting *a pattern or feature vector* and **Y** denoting the associated *class label* of $\underline{\mathbf{X}}$. The components of $\underline{\mathbf{X}}$ are the features.

Definitions :

12) $\underline{\mathbf{X}}$ is called an *ordered or numerical* (see Breiman et.al. [7]) pattern if its features take values from an ordered set, and *categorical* if its features take values from a set not having a natural ordering. Ordered or numerical features can have either discrete or continuous values. For simplicity of notation let us assume that $\underline{\mathbf{X}}$ is of continuous ordered type; furthermore, let $\underline{\mathbf{X}}$ take values from R^q. Let **Y** take integer values {1,2, ..., J}; i.e., there are J classes of concern. Then the goal of any classification scheme in general, and DTC in particular, is to estimate **Y** based on observing $\underline{\mathbf{X}}$.

13) A decision rule d(.) is a function that maps R^q into {1,2, ..., J} with $d(\underline{X})$ representing the class label of feature vector $\underline{X}$. The *true misclassification rate* of d, denoted by $R^*(d)$, is

$$R^*(d) = p(d(\underline{X}) \neq Y)$$

where p(.) denotes the probability.

Let us denote the available labeled samples as $\mathcal{L}=\{(\underline{X}_n, Y_n), n=1,2, \ldots, N\}$.

14) Usually the available labeled sample are divided into two parts: the training sample $\mathcal{L}^{(1)}$ and the test sample $\mathcal{L}^{(2)}$. Commonly $\mathcal{L}^{(1)}$ is taken to be randomly sampled 2/3 of $\mathcal{L}$ and $\mathcal{L}^{(2)}$ the remaining 1/3 of $\mathcal{L}$.

15) Due to the difficulty of computing the true misclassification rate $R^*(d)$, it is usually estimated from either the training set or the test set. Let us denote the estimated misclassification rate as R(d). When the training set is used to estimate $R^*(d)$, R(d) is called the *resubstitution estimate* of $R^*(d)$, and when the test sample is used to estimate $R^*(d)$, R(d) is called a *test sample estimate* of $R^*(d)$. In either case, the misclassification rate is simply estimated by the ratio of samples misclassified to the total number of samples used to do the test. A more complex method of estimating the misclassification rate is the *K-fold cross-validation* method. Here, the data set $\mathcal{L}$ is divided into k nearly equal parts $\mathcal{L}^1, \mathcal{L}^2, \ldots, \mathcal{L}^K$(usually K=10). Then $\mathcal{L} - \mathcal{L}^k$ is used for the training sample, and $\mathcal{L}^k$ is used for the test sample. Next the test sample estimate of the misclassification rate for each k is found, and averaged over K. Note that when K=N (the size of the labeled sample), N-fold cross-validation is also called the "leave-one-out" estimate.

III. POTENTIALS AND PROBLEMS WITH DECISION TREE CLASIFIERS

Decision tree classifiers (DTC's) are attractive for the following reasons [41],[53],[54],[60],[78], [81]

1) Global complex decision regions (especially in high-dimensional spaces) can be approximated by the union of simpler local decision regions at various levels of the tree.

2) In contrast to conventional single-stage classifiers where each data sample is tested against all classes, thereby reducing efficiency, in a tree classifier a sample is tested against only certain subsets of classes, thus eliminating unnecessary computations.

3) In single stage classifiers, only one subset of features is used for discriminating among all classes. This feature subset is usually selected by a globally optimal criterion, such as maximum average interclass separability [78]. In decision tree classifiers, on the other hand, one has the flexibility of choosing different subsets of features at different non-terminal nodes of the tree such that the feature subset chosen optimally discriminates among the classes in that node. This flexibility may actually provide performance improvement over a single-stage classifier [97].

4) In multivariate analysis, with large numbers of features and classes, one usually needs to estimate either high-dimensional distributions (possibly multi-modal) or certain parameters of class distributions, such as a priori probabilities, from a given small sized training data set. In so doing, one usually faces the problem of "high-dimensionality." This problem may be avoided in a DTC by using a smaller number of features at each non-terminal node without excessive degradation in the performance.

The possible drawbacks of DTC, on the other hand, are:

1) Overlap (for the definition, see Sec. II) especially when the number of classes is large, can cause the number of terminals to be much larger than the number of actual classes and thus increase the search time and memory space requirements. Possible solutions to this problem will be addressed in Sec. IV.

2) Errors may accumulate from level to level in a large tree. It is pointed out by Wu et.al.[95] that one cannot simultaneously optimize both the accuracy and the efficiency; for any given accuracy a bound on efficiency must be satisfied.

3) Finally, there may be difficulties involved in designing an optimal DTC. The performance of a DTC strongly depends on how well the tree is designed.

IV. DESIGN OF A DECISION TREE CLASSIFIER

The main objectives of decision tree classifiers are: 1) to classify correctly as much of the training sample as possible; 2) generalize beyond the training sample so that unseen samples could be classified with as high of an accuracy as possible; 3) be easy to update as more training sample becomes available (i.e., be incremental - see section IV) ; 4) and have as simple a structure as possible. Then the design of a DTC can be decomposed into following tasks [46],[50],[51]

1) The appropriate choice of the tree structure.
2) The choice of feature subsets to be used at each internal node.
3) The choice of the decision rule or strategy to be used at each internal node.

When a Bayes point of view is pursued, the optimal tree design may be posed as the following optimization problem

$$\underset{T,F,d}{\text{Minimize}} \quad p_e(T, F, d) \tag{3.1}$$

Subject to: Limited training sample size

where P_e is the overall probability of error, T is a specific choice of the tree structure, F and d are the feature subsets and decision rules to be used at the internal nodes, respectively. The implication of the above constraint is that,with a limited training sample size, the accuracy of the estimates of class conditional densities may deteriorate as the number of features increases. This is also known as the Hughes phenomena [39]. In practice this places a limit on the number of features that may be utilized. That is out of, say L, available features, one forms a new feature set of size N, e.g., by subset selection or by combining features; where usually N << L. Note that for a fixed size N, selected as the best feature subset to discriminate among *all* the classes, the minimum probability of error decision rule, by definition, is the single-stage Bayes rule. But, if one is allowed to select different feature subsets for differentiating between different groups of classes (i.e., a tree structure), one may be able to obtain even smaller probabilities of error than those predicted by Bayes rule.

The above optimization problem can be solved in two steps [51]:

step 1: for a given T and F, find $d^* = d^*(T,F)$ such that

$$p_e(T, F, d^*(T,F)) = \min_d P_e(T, F, d) \tag{3.2}$$

step 2: Find T^* and F^* such that

$$P_e(T^*, F^*, d^*(T^*, F^*)) = \min_{T,F} p_e(T, F, d^*(T, F)) \tag{3.3}$$

It should be noted here that no mention of time complexity or computation speed has been made so far. Including these factors would make the optimization problem even more difficult. Swain and Hauska [78], and Wang and Suen [81]-[84] have offered some ways to include time efficiency (i.e. speed) into the analysis. These are discussed in the latter part of the paper.

When the information theoretic point of view is pursued, however, the optimal design of the tree involves maximizing the amount of average mutual information gain at each level of the tree. We will return to this point shortly.

IV.1.a. DECISION TREE STRUCTURE DESIGN

Several methods [8],[15],[17],[46],[49],[54],[62],[66],[69],[71],[74],[78],[82],[95]-[96] have been proposed for the tree structure design. Some have no claim of optimality and utilize the available a priori knowledge for the design [4],[30],[53],[69],[74],[81],[95],[97], while others apply mathematical programming methods such as dynamic programming or branch-and-bound techniques [46],[47],[49],[58],[62]. Some studies [54],[66],[72] have attempted to combine the tree structure design task with feature selection and the decision rules to be used throughout the tree.

Some of the common optimality criteria for tree design are: minimum error rate, min-max path length, minimum number of nodes in the tree, minimum expected path length, and maximum average mutual information gain. The optimal tree is constructed recursively through the application of various mathematical programming techniques such as dynamic programming [57] with look ahead (or back) capabilities to approach global optimality. A basic problem with any of the suggested optimization methods is their computational infeasibility; usually large amounts of computational time and storage (space) are required. This in practice has placed a restriction on the type of tree and/or the number of features (variables) to be used at each node [97].

In many pattern recognition problems at the outset, one has only a set of labeled samples $\mathcal{L}$ from the set of possible classes. The problem is then, given $\mathcal{L}$, find a decision tree T that optimizes some cost function, such as the average number of nodes in the tree. However, there may be several ways the labeled sample space could be partitioned, all being equally attractive and acceptable. For instance, Figure 3 shows two possible binary partitionings using hyperplanes perpendicular to the feature axes of the space with 3 classes and 2 features; both provide 100% correct classification of the labeled samples.

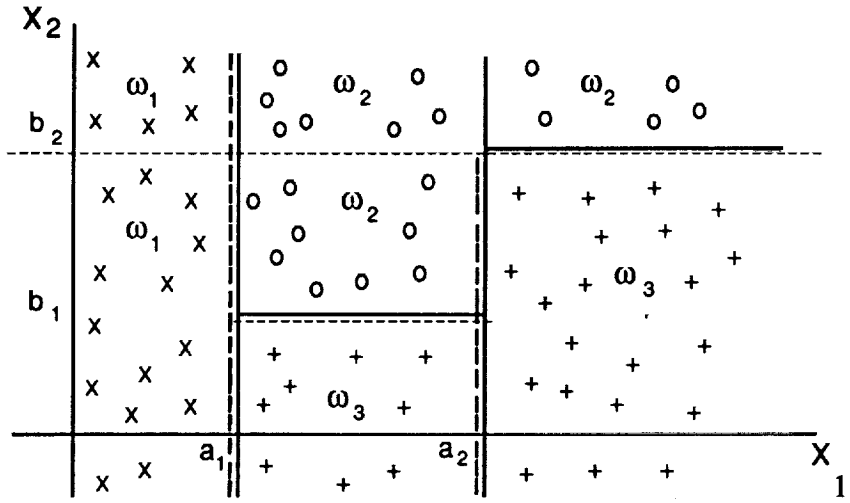
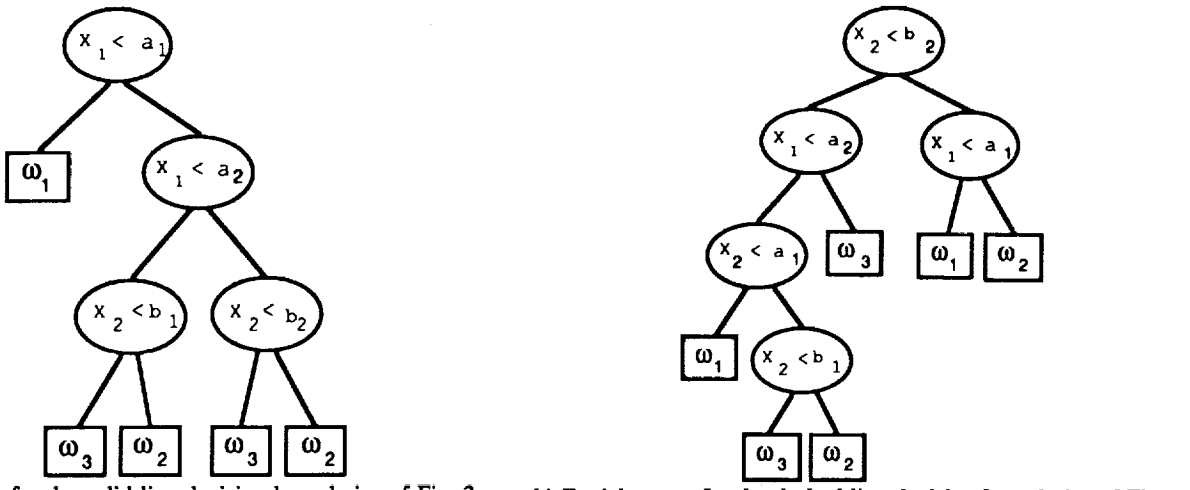


Figure 3. Two dimensional feature space showing two possible partitionings of the same space, each providing 100% classification accuracy. Samples from the three classes are designated x, o, and +. The two sets of decision boundaries are designated by the solid lines and the dashed lines.

For each partitioning, there is a corresponding binary tree. Figure 4 shows the binary trees corresponding to the partitioning of Figure 3. The goal, then, is to construct a binary tree that is equivalent in the sense of giving the same decision, but optimal in the sense of having the minimum average number of nodes, or more generally satisfying some size optimality criterion. Meisel and Michalopoulos [58], and Payne and Meisel [62] address exactly this problem.



a) Decision tree for the solid line decision boundaries of Fig. 3. b) Decision tree for the dashed line decision boundaries of Fig. 3.

Figure 4. Decision trees for the feature space partitions of Figure 3.

The algorithms they provide are basically composed of two parts. In the first part, they find a sufficient partitioning of the space; there are actually many ways to do this part [24],[45],[62]; some provide suboptimal partitioning [45]. A general approach [62] is to sequentially examine the univariate projections of the sample onto each feature axes, then find the hyperplanes perpendicular to the feature axes that partitions the projected samples as accurately as possible, with 100% accuracy if samples are linearly separable. These hyperplanes partition the space into hypercuboids. Also then the problem is transformed to one with the discrete features. Each hypercuboid is labeled with the class label of the majority of the samples in that subregion. The hyperplanes are extended to infinity to create a regular lattice, L, thus each lattice point has a class label and a feature subset associated with it. See Figure 5. Lattice L can be binary partitioned on the feature axis i into mutually exclusive left and right sets, where a left (right) set includes lattice elements with feature i values smaller (larger) than the threshold value. A binary partition can also be denoted by a 5-tuple (n, i , m, n^L, n^R), where n denotes a decision node label for the partition, i and m are the feature axis and the threshold value used for the partition, and n^L and n^R are the node labels for partition of the left and right sets, respectively. Note that a binary tree is a collection of *nested* binary partitions, and can conveniently be represented in the following *recursive* form

$$T = \{ (n, i, m, n^L, n^R) , T^L, T^R \} \qquad (3.4)$$

starting from the root; where T^L and T^R denote the subtrees defined on the left and right sets of a partition.

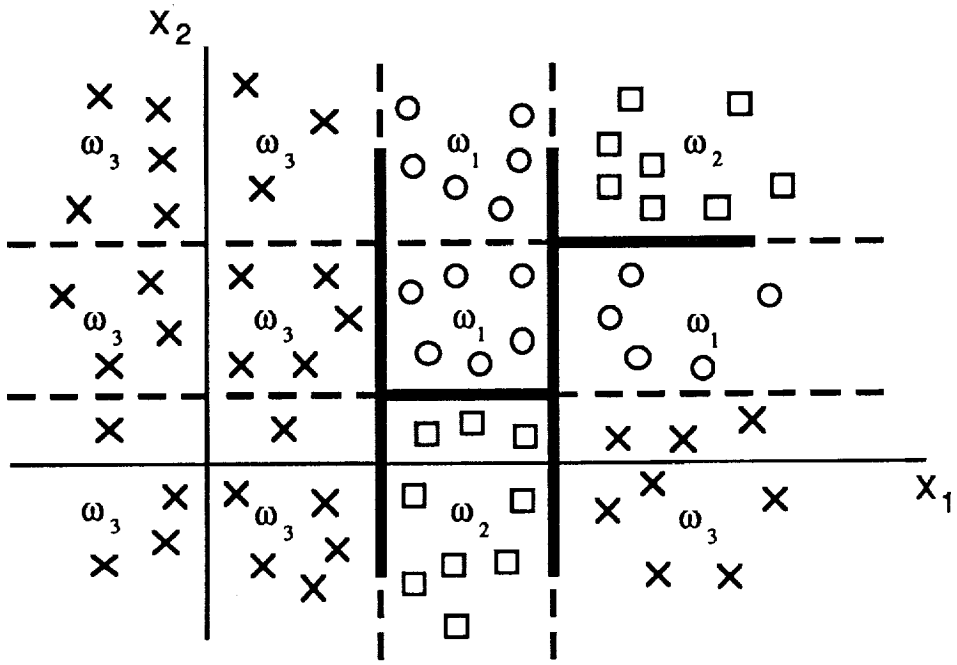


Figure 5. A regular lattice constructed by extending the original partitioning, with each element of the lattice assigned a class label.

For the binary trees defined on this lattice L, the general cost function considered has the form [62]

$$C(T) = F\big(L, C(T^L), C(T^R)\big) \tag{3.5}$$

where $F(L, . , . ,)$ is component-wise non-decreasing and $F \geq 0$. The optimization problem, is then given L, find T^* such that

$$C^*(L) = \underset{T}{\text{Min}}\, C(T) = C(T^*) \tag{3.6}$$

The optimal tree, therefore, could be recursively constructed through the application of invariant embedding (dynamic programing).

Remarks:

1) Only one feature is examined at each node.

2) The algorithms are feasible only for a small number of features; else the size of the lattice becomes large and storage space requirements become a problem. This is because during the optimization process intermediate results must be fully accessible. This perhaps is the main limitation.

The problem of designing a truly optimal DTC seems to be a very difficult problem. In fact it has been shown by Hyafil and Rivest [40] that the problem of constructing optimal binary trees,

optimal in the sense of minimizing the expected number of tests required to classify an unknown sample is an NP-complete problem and thus very unlikely of non-polynomial time complexity. It is conjectured that more general problems, i.e. the problem with a general cost function or minimizing the maximum number of tests(instead of average) to classify an unknown sample would also be NP-complete. They also conjecture that no sufficient algorithm exists(on the supposition that $P \neq NP$) and thereby supply motivation for finding efficient *heuristics* for constructing near-optimal decision trees.

The various heuristic methods for construction of DTC can roughly be divided into four categories: The *Bottom-Up* approaches, the *Top-Down* approaches, the *Hybrid* approach and the tree *Growing-Pruning* approach . In a bottom-up approach [53], a binary tree is constructed using the training set. Using some distance measure, such as Mahalanobis-distance, pair-wise distances between a priori defined classes are computed and in each step the two classes with the smaller distance are merged to form a new group. The mean vector and the covariance matrix for each group is computed from the training samples of classes in that group, and the process is repeated until one is left with one group at the root. This has some of the characteristics of an unsupervised hierarchical clustering approach. In a tree constructed this way, the more obvious discriminations are done first, near the root, and more subtle ones at later stages of the tree. It is also recommended [53] that from a processing speed point of view, the tree should be constructed such that the most frequently occurring classes are recognized first.

It is worth noting that usually most decision tree designs are restricted to having a binary structure. This is not really a restriction since any *ordered* tree can be uniquely transformed into an *equivalent* binary tree [66].

In the top-down approach, the design of DTC reduces to the following three tasks:

1) The selection of splits.
2) The decision as to which nodes are terminal.
3) The assignment of each terminal node to a class label.

Of the above three tasks, the class assignment problem is by far the easiest. Basically, to minimize the misclassification rate, terminal nodes are assigned to the classes which have the highest probabilities. These probabilities are usually estimated by the ratio of samples from each class at that specific terminal node to the total number of samples at that specific terminal node. Then this is

just the basic majority rule; i.e., assign to the terminal node the label of the class that has most samples at that terminal node.

It seems that most of the research in the DTC design has concentrated in the area of finding various *splitting rules*; this also naturally embodies the *termination rules*. Following is a summary of some of the research done in this direction.

Wu et. al [95] have suggested that a histogram based on an interactive top-down approach for the tree design may be especially useful for remote sensing applications. In this approach, the histogram of the training data of all classes is plotted on each feature axis with the same scale. By observing the histograms, a threshold is selected to partition those classes into several groups. If a group contains more than one class, the same procedure is repeated until each group contains only one class. The drawback of this method is that only a few features (usually one) are used at each stage. Interaction between features cannot be observed.

When the distributions of classes under consideration are known, You and Fu [97] proposed the following approach for design of a *linear* binary tree classifier. Since even for moderately small numbers of features and classes, the number of possible trees is astronomically large, they suggested two restrictions to reduce the size of the search space. First, limit the number of features to be used at each stage. Secondly, for the sake of accuracy, specify tolerable error probabilities at each stage. Obviously the choice of linear classifiers and a binary tree structure is made to decrease computational complexity and time and thus to increase the speed. Again, using a distance measure, such as Bhattacharyya distance, classes at each node are divided into two groups. Then, using an iterative procedure with an initial guess, a classifier is found that provides minimum probability of error. If this error exceeds the pre-assigned error bound, the class that commits the maximum error is taken from consideration (i.e. included in both of the following subgroups causing overlap) and, repeating the above procedure, a new classifier is found and the error assessed. Presuming that now the error is below the specified error bound, new subgroups are formed, else the entire above process is repeated. Assuming that the error calculation function is first order differentiable with respect to the coefficients of the linear equations of the classifier, they recommended the Fletcher-Powell algorithm [19].

Remark: Even though allowing overlap is one way of improving recognition rate, caution must be exercised since overuse of it will cause the number of terminals to be much larger than the actual number of classes, thus reducing the efficiency of the tree classifier. This could be a serious drawback in *large-class* problems.

A heuristic procedure which incorporates the time component in the tree structure design is proposed by Swain and Hauska [78], and Wu et.al. [95].For every node n_i an evaluation function $E(n_i)$ is defined as

$$E(n_i) = -T(n_i) - w\,e(n_i) + \sum_{j=1}^{C_i} p_{i+j}\,.\,E(n_{i+j}) \tag{3.7}$$

where $T(n_i)$ and $e(n_i)$ are the computation time and the classification error associated with node n_i, respectively. w is the weighting factor, specified by the user, reflecting the relative importance of accuracy to the computation time. And C_i is the number of descendent nodes of n_i. The third term on the right side of the above equation is the sum of the evaluation functions at the descendent nodes of n_i, weighted by their probabilities of access from n_i. Obviously in this forward top-down search procedure, the configuration of the descendent nodes of n_i are not known yet. Their evaluation function values, however, can be lower-bound-estimated by the values of the corresponding evaluation functions if the usual single-stage (one shot) classifier were used at n_i. The configuration that provides the maximum evaluation function among all the candidate configurations is selected and the process is repeated at the following nodes. As pointed out in [78], this node-by-node design approach, using only local (not global) information, can provide at best only a suboptimal result.

Since in top-down approaches to tree design, sets of classes are successively decomposed into smaller subsets of classes, some researchers have attempted to combine the tree design problem and feature selection problem by using an appropriate splitting criteria at each non-terminal node [54],[66],[72]. Rounds [66] has suggested Kolmogorov-Smirnov distance and test as the splitting criteria. Considering a two-class problem with a binary tree structure in mind, only one feature is selected at each non-terminal node, with the corresponding threshold value. The rationale of this approach is that crossover points of probability distributions of two classes (especially when classes have multi-modal distributions) are good locations to place the partitioning planes. These crossover points also correspond to the relative maxima of difference of cumulative distributions, also known as the Kolmogorov-Smirnov (K-S) distance. In the absence of true class distributions, empirical distributions are estimated from the training set. The K-S distance being a random-variable, a measure of confidence is provided by the Kolmogorov-Smirnov test. At each node for each feature, the K-S distances and confidence level are estimated. The actual value of the feature that provides above two maxima is used as the threshold value to create the descendant nodes.

This approach provides a means for evaluating the discrimination power and significance of each feature in the classification process but has the following shortcomings. First of all, it requires as many decision trees to be developed as there are pattern classes [72]. Second, this method uses only one feature at each node. Third, even for one feature and for a two-class problem, it results in a tree with differentiated structure and a large number of nodes. Furthermore, the K-S distance depends on the sample sizes of both classes at each non-terminal node, which would be in general different at the different non-terminal nodes. The distribution of this statistic under certain restrictions on the relative values of sample sizes for each class can be found in Gnedenko and Kareljuk [26] and Rounds [67].

Li and Dubes [54] proposed a permutation statistic as the splitting criterion. The permutation statistic measures the degree of similarity between two binary vectors: the vector of thresholded feature values, obtained from the training data, and the vector of known pattern class labels. The advantages of this splitting criterion over, for instance, a statistic based on distance between empirical distributions (e.g., K-S distance) are its known distribution and its independence from the number of training patterns. Even though the permutation statistic is based on the two-class assumption, multi-class problems can be handled by a sequence of binary decision trees.

In general, the basic idea in choosing *any* splitting criteria at an internal node is to make the data in the son nodes "purer". One way to accomplish this task [7] is to define an *impurity function* i(t) at every internal node t. Then suppose that for the internal node t there is a candidate split S which divides node t into left son t_L and the right son t_R such that a proportion p_L of the cases in t go into t_L and a proportion p_R go into t_R. One could then define the goodness of the split S to be the decrease in impurity

$$\Delta i(S,t) = i(t) - i(t_L)p_L - i(t_R)p_R$$

Then choose a split that minimize $\Delta i(S,t)$ over all splits S in some set $\mathcal{S}$. The impurity function suggested in [7] is the *Gini* index defined as

$$i(t) = \sum_{i \neq j} p(i \mid t)p(j \mid t)$$

where $p(i \mid t)$ is just the probability of a random sample $\underline{X}$ belonging to the the class i, given we are at node t.

The final component in the top-down DTC design is to determine when the splitting should be stopped; i.e., we need a stopping rule. Early approaches to selecting terminal nodes were of the form: set a threshold $\beta > 0$ and declare a node t terminal if

$$\max_{S \in \mathcal{S}} \Delta i(S(t), t) < \beta$$

The problem with this rule is that usually partitioning is halted too soon at some nodes and too late at some others. The other major problem in deciding when to terminate splitting is the following. Suppose we have classes that their underlying distributions overlap; that is the true misclassification rate is positive ($R^*(d) > 0$) . Furthermore, suppose that we have a training set $\mathcal{L}$ and we use a splitting approach (e.g. impurity function) along with a terminal labeling approach mentioned earlier, and continue the splitting until every node has pure samples; i.e., samples from only one class. Now consider the resubstitution estimate (see III, definition 15) of the misclassification rate of the tree T defined as

$$R(T) = \sum_{t \in \tilde{T}} r(t)p(t) = \sum_{t \in \tilde{T}} R(t)$$

where r(t) is the resubstitution estimate of the misclassification, given a sample falls into node t, p(t) is the probability of a random sample falling into node t, and $\tilde{T}$ is the set of terminal nodes of the tree T . Notice that for above scenario R(T) = 0! Obviously this estimate is very biased. That is, in general, R(T) decreases as the number of terminal nodes increases. But the tree so constructed usually fails miserably when the test data is run through them. Breiman, et.al. [7] conjectured that decision tree design is rather insensitive to a variety of splitting rules and it is the stopping rule that is crucial. Breiman et.al. [7] suggest that instead of using stopping rules, continue the splitting until all the terminal nodes are pure or nearly pure, thus resulting in a large tree. Then selectively prune this large tree upward, getting a decreased sequence of subtrees. Finally use cross-validation to pick out the subtree which has the lowest estimated misclassification rate. To go any further, we need the following definitions (see [7] for more detail).

Definitions :

16) A branch T_t of tree T with root node $t \in T$ consists of node t and all the descendants of t in T.
17) Pruning a branch T_t from a tree T consists of deleting from T all descendants of t. Denote the pruned tree as $T-T_t$.

18) If by successively pruning off branches from T we obtain the subtree T', we call T' a pruned subtree of T and denote it by T'< T.

Recall that the *size* of a tree is of utmost importance. In order to include the complexity of tree T in the design process, define a cost-complexity measure $R_\alpha(T)$ as

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}|$$

where R(T) has the usual meaning of estimated misclassification rate of tree T, and $\alpha \geq 0$ is the complexity cost measure . That is the complexity of tree T is measured by the number of its terminal nodes, and α is the complexity cost per (terminal) node. Let us denote the fully grown tree as T_{max}. Then the objective of a pruning process is, for every α , find the subtree $T(\alpha) \leq T_{max}$ such that

$$R_\alpha(T(\alpha)) = \min_{T \leq T_{max}} R_\alpha(T)$$

Notice that as α is varied from sufficiently small values where T_{max} itself is the optimally pruned tree, to sufficiently large values where T_{max} is pruned so far that only the root node {root} is left, one obtains a finite sequence of trees T_1, T_2, ... , {root}; where T_i corresponds to the optimal subtree for $\alpha = \alpha_i$. Thus the pruning algorithm due to Breiman et.al.[7] is

procedure Tree Pruning Algorithm [7]:

begin

comment let T_{max} denote the fully grown tree;

input T_{max} ;

i =1 ;

$\alpha_i = 0$;

comment find T_1, the optimum subtree of T_{max} for $\alpha_1 = 0$;

FIND_ T_1 ;

$T_i = T_1$;

while $T_i \neq$ {root} **do**

begin

comment find the *weakest - link* T_{t_i} in the subtree T_i and its corresponding value of α_i ;

FIND_ T_{t_i} & α_{i+1} ;

comment prune off T_{t_i} from tree T_i to form the optimal tree T_{i+1} ;

$T_{i+1} = T_i - T_{t_i}$;

$i = i + 1$;

end

comment at this point we have a decreasing sequence of subtrees

$T_1 > T_2 > T_3 > \ldots > \{root\}$;

SELECT_ best tree ;

end

procedure FIND_ T_1 :

begin

comment recall that for any node t with children t_L and t_R , $R(t) \geq R(t_L)+R(t_R)$;

for any two terminal nodes t_L and t_R having the same parent t **do**

begin

if $R(t) = R(t_L)+R(t_R)$ **then**

begin

eliminate t_L and t_R ;

label the parent node t as a terminal node ;

end

T_1 = pruned tree ;

end

end

procedure FIND_ T_{t_i} & α_{i+1} ;

begin

comment let $T_t^{(i)}$ denote a subbranch of T_i with root node t and {t} the subbranch consisting of only the node t . Recall

$R_\alpha(\{t\}) = R(t) + \alpha$,

$$R_\alpha(T_t^{(i)}) = \sum_{t' \in \tilde{T}_t^{(i)}} R(t') + \alpha \, | \tilde{T}_t^{(i)} | \ ,$$

$$R_\alpha(\{t\}) \geq R_\alpha(T_t^{(i)}) \ ;$$

comment denote by $\alpha_i(t)$, the value of α at node t for which

$$R_\alpha(\{t\}) = R_\alpha(T_t^{(i)}) \ ;$$

$$\alpha_i(t) = \frac{R(t) - R(T_t^{(i)})}{| \tilde{T}_t^{(i)} | - 1} \ ;$$

comment this means that branch $T_t^{(i)}$ can be replaced with node {t} ;

$$\alpha_{i+1} = \min_{t \in T_i} \alpha_i(t) \ ;$$

comment suppose minimum occurs for $t = \bar{t}_i$ .This means $T_{\bar{t}_i}$ is the *weakest-link* in T_i . That is, as α increases, $T_{\bar{t}_i}$ is the first subtree that can be pruned off ;

end

procedure SELECT_best tree :

begin

use test sample or cross-validation to estimate the misclassification rate of trees

$T_1, T_2, \ldots, \{root\}$;

select the tree T_k with smallest misclassification rate ;

end

Note that the pruning algorithm is essentially a (multipass) top-down algorithm. The tree growing and pruning of Breiman et.al. [7] has been incorporated into a computer program known as CART (Classification and Regression Trees.) Following are some of the shortcomings of CART. First of all, CART allows only either a single feature or a linear combination of features at each internal node. Second, CART is computationally very expensive as it requires generation of multiple auxiliary trees. Finally and perhaps most importantly, CART selects the final pruned subtree from a parametric family of pruned subtrees, and this parametric family may not even include the optimal pruned subtree. There are, of course, other tree growing and, more importantly, pruning

algorithms (e.g. [12],[25],[66]). Gelfand et.al. [25] propose the following *iterative* tree growing and pruning algorithm:

1) Divide the data set into two approximately equal sized subsets and iteratively grow the tree with one subset and prune it with the other subset.

2) Successively interchange the role of the two subsets.

Their pruning algorithm is the following simple and intuitive *one-pass* bottom-up approach which starts from the terminal nodes and proceeds up the tree, pruning away branches.

procedure Tree Pruning Algorithm [25]:
begin
comment let T denote the tree to be pruned and start from the terminals of the tree;
for every node t in the tree T **do**
if $t \in \tilde{T}$ **then** $S_\alpha(t) = R_\alpha(t)$;
else
if $t \in T - \tilde{T}$ **then**
begin
$S_\alpha(t) = S_\alpha(\text{left}(t)) + S_\alpha(\text{right}(t))$;
comment left(t) and right(t) are left and right sons of t, respectively;
if $R_\alpha(t) \leqq S_\alpha(t)$ **then**
begin
$T = T - (T_{\text{left}(t)} \cup T_{\text{right}(t)})$;
left(t) = 0, right(t) = 0, i.e., prune off the left and right sons;
$S_\alpha(t) = R_\alpha(t)$;
end
end
end

They prove the convergence of their algorithm and their experimental results on waveform recognition, along with the theoretical proof, suggests the superiority of their algorithm to the pruning algorithm of Breiman et.al.[7].

Recently, a hybrid method was proposed by Kim and Landgrebe [42] that uses both bottom-up and top-down approaches sequentially. The rational for the proposed method is that in a top-down approach such as hierarchical clustering of classes, the initial cluster centers and cluster shape information are unknown. It is also well known that the proper choice of initial conditions could considerably influence the performance of a clustering algorithm (e.g., speed of convergence and final clusters). This information can be provided by a bottom-up approach. Then the algorithm for construction of a binary tree is as follows:

procedure Hybrid Tree Design [42] :
begin
    root = set of all classes;
    comment test root to see if it has only one class ;
    TEST(root) ;
end
Procedure TEST(root) :
begin
    If | root | = 1 **then** label root as a terminal
    else
        begin
            comment use a bottom-up approach to find information about
                two subgroups (clusters) (i.e., find (M_i , Σ_i) i =1,2);
            BOTTOM-UP (root) ;
            comment use a top-down approach to cluster classes in the root node
                into 2 clusters;
            TOP-DOWN (root, cluster 1, cluster 2) ;
        end
end
procedure BOTTOM-UP (root):
begin
        Use any bottom-up approach (e.g., [53]) to come up with two subgroups
        combinations of which would result in the root node;
        Compute the mean M_i and covariance Σ_i of these two subgroups;
end
procedure TOP-DOWN (root, cluster 1, cluster 2):
begin

Use the means of the subgroups computed above, as the cluster centers;
Cluster the entire set of classes at the root to two groups using the normalized sum of square error criterion:

$$\sum_{i=1}^{2} \sum_{x \in C_i} [\, (x - M_i)^T \Sigma_i^{-1} (x - M_i) + \ln |\Sigma_i| \,]$$

where C_i is the cluster i, and M_i and Σ_i are as computed above;

for $i \leftarrow 1$ until 2 **do**

If cluster i has only one element **then** label it as a terminal;

else

begin

root = C_i ;

TEST (root);

end

end

In all the foregoing discussions, at each node a decision is made and only one path is traversed from the root of the tree to a terminal node. This is referred to as a "hard-decision system" by Schuerman and Doster [69]. In contrast to hard-decision systems, they propose an approach where all the a posteriori probabilities are approximated at each non-terminal node without making a decision at any of those points; a decision is made only at the terminal nodes of the tree by selecting the maximum a posteriori probability. In other words, in a manner similar to single-stage approaches [23], a posteriori probabilities for the different classes are estimated, but in a sequence of steps. Obviously, one would expect to obtain the same a posteriori probabilities at the terminal nodes as the ones the global single stage classifier would give, assuming the estimated a posteriori probabilities at each non-terminal node are the same as the actual values.

In single-stage classifications, one way to classify an unknown sample into one of m classes is to evaluate $(m-1)$ discriminant functions [18],[23]. When Bayes' minimum probability of error is the desired criterion, these discriminant functions are the a posteriori probabilities. In a sequential approach to evaluate these probabilities, the conditional probabilities of classes at each of the output branches of a non-terminal node are estimated, based on the set of classes and the full feature measurement vector at the input to that node. Obviously the structure of the tree and the accuracy of these estimates are somewhat related. But, clearly, if the true a posteriori probabilities could be furnished, the explicit structure of the tree would not be critical. So, at least conceptually, the input

classes at each node can be decomposed in a way that a posteriori probabilities of the classes at the outgoing branches may be estimated with "maximal confidence." Again due to the enormous number of possible combinations at each node, the actual tree is usually constructed using training data and heuristic methods based on a separability measure.

Mean square polynomial discriminant functions, using training sets, are suggested [96] for estimating the a posteriori probabilities of classes at the output of each node branch given the set of classes at the input of the node and the full feature measurement vector.

Remarks:

1) By thresholding these a posteriori probabilities at each node, if desired, the soft-decision approach can be converted to the conventional hard-decision approach.

2) When the number of classes is large and no overlap is allowed in the tree, the hard-decision approach raises the possibility of error accumulation [81]-[83]. Even though using some heuristics this error can be reasonably maintained [67], but the soft-decision approach avoids the problem altogether.

3) When the number of classes and features are large and the class distributions have significant overlap, a hard-decision approach covers up ambiguities by a "forced recognition" [69]. In the soft-decision approach, no decision is made until the last stage of the tree.

4) In general, however, the computational time complexities of the soft-decision approach may be a limiting factor.

IV.1.b. ENTROPY REDUCTION AND INFORMATION-THEORETIC APPROACHES

A different point of view about pattern recognition is taken by Watanabe [76],[85]-[93]. Since organization and structure of objects or a set of stochastic variables could be expressed in terms of entropy functions [86], he refers to the problem of pattern recognition as that of "seeing a form" or structure in an object [86], and he suggests ways to cast the problems of learning [88],[89], such as feature (variable) selection, dimensionality reduction [91], and clustering [90],[92], to mention a few, in terms of minimizing properly defined entropy functions. As noted by Suen and Wang [77],[81] this point of view could be very attractive when the number of classes is large and

calculation of Bayes' error probabilities is not so simple. Basically, since at the root of a tree, a given sample could belong to any of the classes, the uncertainty is maximum. At the terminal nodes of the tree, the sample is eventually classified and the uncertainty is eliminated. So, an objective function for a tree design could be to minimize uncertainty from each level to the next level, or in other words, maximize entropy reduction at each stage. Since it is also desirable to have as few overlaps as possible, Suen and Wang [77] suggested an interactive iterative clustering algorithm (ISOETRP) with an objective function directly proportional to the entropy reduction and inversely proportional to some function of class overlap at each stage of the tree. Even though any entropy measure could be used, Shannon's entropy, defined as

$$H = \sum_i p_i \log p_i \tag{3.8}$$

where p_i the a priori probability of class i, is preferred because of its strong additivity property [1].

Sethi and Sarvarayudu [74] propose a simple, yet elegant, method for the hierarchical partitioning of the feature space. The method is non-parametric and based on the concept of average mutual information. More specifically, let the average mutual information obtained about a set of classes C_k from the observation of an event X_k, at a node k in a tree T be defined as

$$I_k(C_k ; X_k) = \sum_{C_k} \sum_{X_k} p(C_{ki}, X_{kj}) \cdot \log_2 \left[\frac{p(C_{ki} \mid X_{kj})}{p(C_{ki})} \right] \tag{3.9}$$

Event X_k represents the measurement value of a feature selected at node k and has two possible outcomes; measurement values greater or smaller than a threshold associated with that feature at that node.

Then, the average mutual information between the entire set of classes, C, and the partitioning tree, T, can be expressed as

$$I(C ; T) = \sum_{k=1}^{L} p_k I_k(C_k ; X_k) \tag{3.10}$$

where p_k is the probability of the class set C_k and L is the number of internal nodes in the tree T.

Since the probability of misclassification, p_e, of a decision tree classifier T and the average mutual information $I(C ; T)$ are related as [74]

$$I(C;T) \geqq -\sum_{j=1}^{m} [\, p(c_j) \cdot \log_2 p(c_j) \,] + p_e \cdot \log_2 p_e + (1-p_e) \cdot \log_2 (1-p_e) + p_e \cdot \log_2 (m-1) \tag{3.11}$$

with equality corresponding to the minimum required average mutual information for a pre-specified probability of error. Then a goal for design of the tree could be to *maximize* the average mutual information gain (AMIG) at *each* node k. The procedure is best explained by the following algorithm.

procedure AMIG Tree Design [74] :
begin

compute I_{min} for the desired p_e (see eq. 3.11);
$I(C;T) \leftarrow 0$;
continue = ' True' ;
while $I(C;T) < I_{min}$ & continue= 'True' **do**
begin
choose feature j and its corresponding threshold α_j that maximizes
$p_k I(C_k ; X_k)$ (see eq. 3.9);
update $I(C;T)$ (see eq. 3.10);
if ($I(C;T)$ - old $I(C;T)$ > threshold) **then** continue = 'True'
else
continue = 'False' ;
end
end

A tree constructed by this top-down algorithm would have a binary structure, and since at each node the partitioning hyperplane (i.e., the optimum feature and its corresponding threshold) is selected to maximize the average mutual information at that node, the performance is optimized in a "local sense". Once the partitioning is specified, the globally optimum tree could be obtained using the algorithm suggested by Payne and Meisel [62].

Chou and Gray [11] view decision trees as variable-length encoder/decoder pairs, and compare the performance of the decision trees to the theoretical limits given by the rate-distortion function. More specifically, they show that rate is equivalent to the average tree depth while the distortion is

the probability of misclassification. They also use greedy algorithms to design trees and compare their performances to certain classes of optimal trees under certain restrictive conditions. The applicability of their method is limited, in practice, to only small problems. Goodman and Smyth [30] prove that a tree designed based on a top-down mutual information algorithm is equivalent to a form of Shannon-Fano prefix coding.

When the measurement variables (features) are discrete, as for instance, in computer diagnostics of diseases or in stroke directions or number of line crossings in character recognition, the conventional classifier design or feature selection techniques face practical difficulties [72]. This is mainly due to the non-metric nature of the measurement space. Sethi and Chatterjee [72] use concepts of *prime events* to come up with an efficient approach to the decision tree design. Even though their method does not guarantee optimality, in most cases it provides close to optimal results, but most importantly, it is efficient.

IV.2. FEATURE SELECTION IN DTC'S

Regardless of its importance, the problem of feature subset selection in DTC's has received relatively little attention. Kulkarni and Kanal [47] proposed the use of a branch-and-bound technique to assign features to the nodes in a sequential top-down manner. The probability of error is calculated for each possible feature subset selected; if it exceeds the pre-assigned lower bound then that feature subset is abandoned.

Due to computation time, as a practical matter, the size of the feature subset to be used at each node is usually limited to be much smaller than the total number of available features [83],[97]. Once it has been determined (often heuristically) how many features are to be used at each node, different feature subsets are examined and the one that provides maximum separability, measured by some distance function, among classes accessible from that node, is usually selected. Note that when the number of classes is large, exhaustive search may be infeasible. In that case, usually a greedy-type search algorithm is used. Again, this at best would provide local optimality. Some studies, in order to determine the significance of each feature in discriminating among classes associated with each node, have offered methods that utilizes only one feature at each node. It should be noted that while the use of single feature decisions at every internal node reduces the computational burden at the tree design time, it usually leads to larger trees.

Recently, the subject of neural networks, has become very popular in many areas such as signal processing and pattern recognition. Actually, the applications of neural networks in pattern recognition problems go back to the days of simple perceptrons in the 1950's. Many advantages for neural networks have been cited in the literature (c.f. Rumelhart et.al. [68] and Lippmann [56]). Most important of them, for the pattern recognition problems, seems to be the fact that neural network-based approaches are usually nonparametric even though statistical information could be possibly incorporated to improve their performance (speed of convergence, etc.). Also neural networks can extract nonlinear combinations of features, and the resulting discriminating surfaces can be very complex. These characteristics of neural networks can be very attractive in a decision tree classifier where one has to determine the appropriate feature subsets *and* the decision rules at each internal node. There are various neural network models such as Hopfield nets, the Boltzmann machine and Kohonen self-organizing feature maps, to name a few; the most popular network by far, however, is the multilayer feedforward network [68].

A multilayer feedforward neural network is simply an acyclic directed graph consisting of several layers of simple processing elements known as neurons. The neurons in every layer are fully connected to the neurons in the proceeding layer. Every neuron sums its *weighted* inputs and passes it through some kind of nonlinearity, usually a sigmoidal function. The first layer is the *input layer*, the last layer is the *output layer* and the intermediate layers are known as the *hidden layers*. See Figure 6. Then for a supervised learning, training samples are assigned to the inputs of the network, and the connection weights between neurons are adjusted such that the error between the actual outputs of the network and the desired outputs are minimized. Commonly used error criterion is the total sum-of-squared error between the actual outputs and the desired outputs. There are many learning algorithms to perform this learning task; a frequently used learning rule is the back propagation algorithm which is basically a (noisy or stochastic) gradient descent optimization method. Note that, once one fixes the structure of the network, i.e., the number of hidden layers and the number of neurons in each hidden layer are chosen, then the network adjust its weights via the learning rule until the optimum weights are obtained. The corresponding weights (along with the structure of the network) create the decision boundaries in the feature space. The question of how to choose the structure of the network is beyond the scope of this paper and is a current research issue in neural networks. It suffices to mention that for most practical applications, networks with only one hidden layer are utilized. Also see comments in section V and [56].

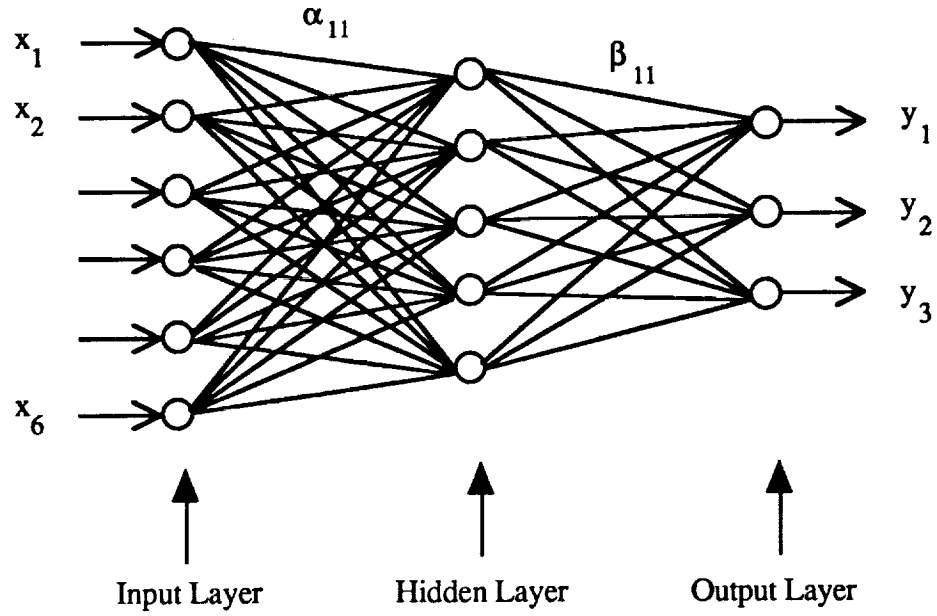


Figure 6. A three layer feedforward network with 6 inputs and 3 outputs. α_{ij} and β_{kl} are the connection weights between neuron i of input layer and neuron j of hidden layer and neuron k of hidden layer and neuron l of output layer, respectively.

How can neural networks assist us in the design of decision trees? Gelfand and Guo [31] propose the following. Recall that in every internal node t of the tree T one would like to find the optimum (here, in the local sense) feature subset and the corresponding decision surface to partition C(t), the set of classes in t, into $C(t_L)$ and $C(t_R)$, two subset of classes at nodes t_L and t_R, respectively. That is there are two nested optimization tasks involved here. The outer optimization loop searches for the optimum partitioning of classes into two (possibly disjoint) subsets of classes and the inner optimization loop searches for the optimum decision surface in the feature space to perform the outer optimization task. Suppose for now that we know the partitioning of C(t) into $C(t_L)$ and $C(t_R)$, and are only looking for the (optimum) decision surface in the feature space. This, of course, can be easily implemented by a simple multilayer feedforward network. Then the remaining question is how to partition C(t) into $C(t_L)$ and $C(t_R)$? As mentioned in section IV.1, there are various splitting rules. One such possible rule is the impurity reduction criterion used in CART. That is, one will consider various partitioning of the C(t) into $C(t_L)$ and $C(t_R)$, and choose the partitioning that gives the maximum impurity reduction (see section IV.1.a, and [7] for details).

Of course, when the number of classes is large, an exhaustive search may be impractical. In this case, some type of greedy search can be performed.

IV.3. DECISION RULES AND SEARCH STRATEGIES IN DTC's

Once the tree structure is designed and the feature subsets to be used at each node of the tree are selected, a decision rule is needed for each node. These decision rules could be designed such that optimal performance (in any specified sense) could be attained at each node (i.e. local optimality); or the overall performance of the tree could be optimized (i.e. global optimality). Obviously, decision strategies designed to provide optimum performance at each node, do not necessarily provide overall optimum performance. For globally optimum performance, decisions made at each node should "emphasize the decision which leads to a greater joint probability of correct classification at the next level" [51]; i.e., decisions made at the different nodes are not independent. Kurzynski [50],[51] addresses both locally and globally optimum decision rules. He shows that the decision rule that provides minimum probability of error at each node (i.e., local optimality) is just the well known maximum a posteriori probability rule. Since for globally optimum rules, error recognition information in the future nodes are needed, the problem is worked in a sequentially backward (bottom up) manner starting from the terminal nodes.

Let us assume that there are m pattern classes and λ_{ij} represents the cost or losses incurred in classifying a sample from class C_i to class C_j. Then the Bayes minimum risk classifier allocates an unknown pattern sample to class C_k if

$$C_k = \arg\left(\min_j \sum_{i=1}^{m} p(C_i \mid X) \lambda_{ij} \right) \tag{3.13}$$

where X is the feature measurement vector of the unknown sample. For a "0-1" loss function, $\lambda_{ij} = 1-\delta_{ij}$ where δ_{ij} is the Kronecker delta

$$\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i\neq j \end{cases} \tag{3.14}$$

and above rule becomes the simple maximum *a posteriori* rule, i.e.,

$$C_k = \arg (\max_i \; p(C_i \mid X)). \tag{3.15}$$

Even though "0-1" is a reasonable and frequently used loss function, in a decision tree classifier, it should only be applied at the terminal nodes where the actual classification task is performed. At

the intermediate nodes in a tree, *decisions* are made as to which of the possible branches leaving that node should be followed. Let α_{ij} represent the cost incurred when a sample of class C_i is routed through branch j. Obviously, α_{ij} is a *random variable* since (Dattatreya and Kanal [13]) actions at the levels below the present level are not known. Dattatreya and Kanal [13] propose an interesting *unsupervised* scheme that *adaptively* learns the *mean* values of these random variables and improves the tree performance.

The only cost considered above was the cost of decision-making and classification. What about the cost of feature measurements? In many applications, such as medical diagnosis, feature measurement costs (e.g., lab tests, X-rays, and etc.) may be a major portion of the cost. Thus more generally, the *total* cost should be (Dattatreya and Sarma [16], and Kulkarni and Kanal [46]) the sum of the feature measurement cost and the classification cost. Assuming that feature measurement costs are constant, Wald [80] shows that the minimum cost classifier is a multistage scheme which measures only one feature at every stage and decides the next course of action. Furthermore, Dattatreya and Sarma [16] show that rejection of class labels at intermediate stages as unlikely candidates is suboptimal in the Bayes minimum cost sense.

Now consider the situation where the tree is designed and feature subsets to be used at each node are determined. In this case, the problem of hierarchical classification can also be viewed as a problem of tree search. Kanal [41], and Kulkarni and Kanal [47] have generalized the idea of state space representation and ordered admissible search methods of Artificial Intelligence (Nilsson [61]) to the problem of hierarchical classification. Then, to search through the tree, an evaluation function $f(n)$ (Kanal [41]) is defined at each node n as

$$f(n) = g(n) + h(n) + l(n) \tag{3.16}$$

where $g(n)$ is the cost from the root to node n, $h(n)$ is (an estimate of) the cost from n to a terminal node *accessible* from n, and $l(n)$ is (an estimate of) the risk of classification at a terminal node accessible from n.

At a goal node, s^*, the total cost is the sum of measurement costs at each node along the path from the root to s^* plus the actual risk, $r(s^*)$, associated with s^*. If $r(s^*)$ is estimated based on only the feature measurements at each node along the path from the root to s^*, the strategy that provides minimum total cost is called (Kanal [41]) an *S-admissible* search strategy. When $r(s^*)$ is estimated using all the measurements, not just those on the path to s^*, then the strategy that provides minimum total cost is known as a *B-admissible* search strategy (Kanal [41]).

At every node, a lower bound on the value of risk is used in the evaluation function for the S-admissible search, except for a goal node where the actual risk is used. In a B-admissible search, however, since the risk associated with any goal s^* can change as additional measurements are made along other paths in the tree, a search could not be terminated simply as one goal state is reached. Therefore, an upper bound function on the risk is also needed to "determine the termination point" (Kanal [41], and Kulkarni and Kanal [47]). Obviously, the performance of either of the above two search methods is dependent on one's ability to find proper bounds on the goal risk. The advantage of the state-space model as pointed out by Kulkarni and Kanal [47] is that, in contrast to the hierarchical classifier approach where only one path is pursued from the root to a terminal point, here provisions are made to back up and follow other alternative paths in the tree if it is needed. Obviously, however, this would increase the number of nodes visited before a decision is made and thus reduce the efficiency. Also search efficiency is a function of the tightness of the bounds on the risk; and tight bounds are difficult to obtain particularly for some continuous distributions [47]. It is also interesting to note that the way cost functions are usually defined neither the B-admissible nor the S- admissible search has provisions for the time component [81].

As mentioned earlier, when the number of classes is large, i.e. on the order of hundreds or even thousands as is in the Chinese character recognition problem for instance, tree classifiers could provide a considerable amount of time savings. But error has a tendency to accumulate from level to level. This is because in problems with large numbers of classes, the tree has many levels. If e is the minimum error at each node of each level and there are on the average l levels in the tree, the average total error rate of the tree would be on the order of $(l * e)$. As proposed in Wang and Suen [81], there are two ways to reduce the total error: either reduce e or have a better search strategy capable of backing-up and re-routing. In general e can only be reduced by allowing overlaps at non-terminal nodes [95]; but this would increase the number of nodes and terminal nodes, thus greatly increasing the memory space requirements. To improve tree search, however, several heuristic search strategies are proposed [2],[9],[10],[28],[34],[36],[81]-[83]. Chang and Pavlidis [10] proposed a branch-bound- backtracking algorithm. A fuzzy decision function [99], taking values in the interval [0,1], is used to assign the decision values of all the branches going out of a node. The overall decision value of a path is defined to be the product of the decision values of all the branches along that path. The product operator could be either the regular product or the max-min operator [9]. Branches with the largest decision values are followed. At any node, if the total decision value up to that node falls below a pre-specified threshold, the algorithm backtracks to the previous node(s) and follows another path. Of course the speed of the search is related to the threshold value set.

Since the most confusion and error occurs for samples that come from regions near the boundaries between classes (or groups of classes), Wang and Suen [81] propose a two-step search approach. In the first step, following the usual top-down search approach, an unknown sample reaches a terminal node. The degree of similarity of the sample with the class label associated with that node is computed. If this value is above a preset threshold, the sample is classified and the next sample is treated; otherwise, the unknown sample goes through the second search called "fuzzy logic search." As argued in [81], the rationale for fuzzy logic is that somehow the global search information and all the possible terminals (classes) must be recorded. Probabilistic decision values based on Bayes decision regions, although perhaps more precise, are usually much harder to estimate. Fuzzy decision values, however, are both flexible and easier to compute. By adjusting the two thresholds involved in the two steps, the speed of the search could be controlled. This would also provide a trade-off between speed and accuracy of the tree classifier.

V. OTHER TREE REALTED ISSUES

A) Incremental tree design:

With regard to the training samples, there are two possibilities: 1) The *entire* training sample is available at the time of decision tree design; 2) the training sample arrives in a *stream.*

For case 1, once a tree is designed the task is completed and such design algorithms are known as nonincremental algorithms. For the second case, however, one has two options:

a) Whenever the new training samples become available, discard the current tree and construct a replacement tree using the enlarged training set.

b) Or revise the existing tree based on the new available information.

Procedures corresponding to b) are known as *incremental* decision tree design algorithms.

Of course, it would be desirable that the incremental decision tree design algorithms produce the *same* trees as those if all the training samples were available at the time of design. Utgoff [79] has developed one such incremental algorithm.

B) Tree generalization:

In designing DTC, one must always keep in mind that the tree designed is going to be used to classify unseen test samples. With this in mind, one has two options: 1) Design a decision tree that correctly classifies *all* the training samples, also known as a *perfect tree* [63]; and select the *smallest* perfect tree. 2) Or construct a tree that is perhaps imperfect (in the above sense) but has the smallest possible error rate in classification of test samples. In practical pattern recognition tasks, it is usually this second type of tree that is of most interest.

Regardless of which of the above two types of trees one may need, it is usually desirable to keep the size of the tree as small as possible. The reasons for this are: 1) Smaller trees are more efficient both in terms of tree storage requirements and test time requirements; 2) smaller trees tend to generalize better for the unseen samples because they are less sensitive to the statistical irregularities and idiosyncrasies of the training data.

C) Missing value problem:

The missing value problem can occur in either the design phase, or the test phase, or both. In the design step, suppose some of the training sample feature vectors are incomplete; that is some feature elements of some feature vectors which are not recorded or are missing. This can happen, for instance, due to some occasional sensor failures. Similarly for the test samples, some feature values may be missing.

In the design phase, one simple but wasteful method to coupe with this problem is to throw away the incomplete feature vectors. For the test sample, of course, this simple option is not acceptable. Breiman et.al. [7] propose the following solution which is based on the idea of using *surrogate splits*. For the case of simplicity, consider the case of binary splitting at every internal node based on the value of only *one* feature. For the extension of this idea to a linear combination of features, see [7] and [21] . The basic idea is as follows.

In the tree design phase, at node t, find the best split s_m^* on the feature element X_m using all the training samples containing a value of X_m. Then select the split s^* which maximizes the impurity reduction $\Delta i\ (s_m^*, t)$ at node t (see section IV 1.a) .

For the incomplete test sample, if at a node t the best split s^* is not defined because of missing feature element values, proceed as follows. Define a *measure of similarity* between two splits s_i and s_j as $\lambda(s_i; s_j)$. Examine all nonmissing feature elements for the test sample; find that one, say X_m, with split $\tilde{s}_m$, that is most similar to s^*. $\tilde{s}_m$ is called a surrogate split to s^*. Then use $\tilde{s}_m$ at node t to decide to traverse to node t_L or t_R.

D) Robustness of decision tree design:

Since decision trees are often constructed by using just some sets of training samples, it is important to make sure that the design procedure is in some sense robust relative to the presence of "bad" samples or outliers. Of course, one could always edit the training data before application in the tree design. This, however, in many cases may not be feasible for the following reasons.

1) The designer may not be aware of the existence of outliers at the time of the tree design, even though outliers are usually easily detected in a training set; in some cases a thorough examination of data set may be necessary.

2) With a small sample, one may not want to throw away any valuable samples.

E) Decision trees and neural networks:

With regards to the decision trees and neural networks, two positions can be taken.

1) Try to utilize neural networks in the implementation of various aspects of decision trees; e.g., Gelfand and Guo [31] use neural networks in the internal nodes of a decision tree to perform the task of decision boundary selection. In this approach, one is constructing a *tree of neural networks*. See also Golea and Marchand [26]

2) Convert a decision tree into a large neural network. This is the direction Sethi [70]-[71], Sethi and Chatterjee [72] and Sethi and Otten [73] have taken. The main idea here is that, as it was mentioned earlier in section IV.2, in the design of multilayer feedforward networks, the structure of the network, i.e., the number of hidden layers and the number of neurons in each hidden layer is not known in advance, and is often chosen rather heuristically and by trial and error. Method describe in [70], [71], [72] and [73] offers one way to uncover the structure of the network. That

is, they offer a method to implement a pre-designed decision tree via a multilayer feedforward neural network.

VI. CONCLUSIONS AND FINAL REMARKS

Decision tree classifiers show a great deal of potential in many pattern recognition problems such as remotely sensed multispectral data classification, medical diagnosis, speech and character recognition, to mention a few. Perhaps one of the main features of DTC's is the flexibility they provide; for example the capability of using different feature subsets and decision rules at different stages of classification and the capability of trade-offs between classification accuracy and time/space efficiency.

Some of the goals of this review have been to:

1) Bring the disparate issues in decision tree classifiers closer together and perhaps motivate some new ideas;

2) Provide a more unified view of decision tree classifiers;

3) And caution the "casual" users of these methods of the possible "pitfalls" of each method.

Even though optimization techniques such as dynamic programing have been offered by some researchers to define trees that have certain optimal characteristics, a *practical* and *feasible* solution that is truly optimal with respect to the choice of tree structure, feature subsets and decision rule strategies is yet far from realization. This is mainly due to either unavailability of the necessary information (e.g., a priori and class conditional statistics) for the design or to complexities of the proposed methods that limits their usefulness for real applications of moderate or large size. Recognizing the difficulties involved in the design of optimal DTC's, many heuristic methods have been proposed in the literature.

Four basic approaches for tree design were examined.

1) Bottom-up approach where one starts with the information classes and continues combining classes until one is left with a node containing all the classes (i.e., the root).

2) Top-down approach where starting from the root node, using a splitting rule, classes are divided until a stopping criterion is met. The main issues in this approach are: a) choice of splitting criterion; b) stopping rules ; c) labeling the terminal nodes.
3) Hybrid approach where one use a bottom-up procedure to direct and assist a top-down procedure.
4) Tree growing-pruning approach where in order to avoid some difficulties in choosing a stopping rule, one grows the tree to its maximum size where the terminal nodes are pure or almost pure. And then selectively prunes the tree.

As was pointed out earlier, computational efficiency, accuracy of classification and storage space requirements usually seem to be conflicting requirements. This fact may actually assist the designer (user) of a DTC in selecting one possible scheme over the others. The existing methods with their relative time/space complexities, degrees of (sub)optimality and proposed areas of applications are summarized in the following tables. Table 1 provides a summary of some of the tree design methods in terms of the assumptions each approach makes, their performance criterion and some of the specific requirements of each method. Table 2, gives the brief summary of various feature selection methods. Table 3, summarizes the various decision rule and search strategies used in decision trees. And finally, table 4 provides specific references for various applications.

Finally it should be pointed out that since the performance of the DTC depends on how well the tree is designed, special attention should be paid to this phase. In particular, when the number of samples to be classified is large, as is, for example, in the remote sensing applications, the time spent for the design could be well justified.

Table 1- Summary of tree structure design

Ref.	Optimal?	Criterion?	Assumptions	Remarks (by author or alternate ref.)
4	Locally (node-wise)	Prob. of error	Given: a well defined feature transformation	-An automated technique -uses only a priori statistics.
8	No	Max. inf. gain at each node.	Given: a binary image with a priori class prob.'s; and number of black bits in each position in each class in the design samples.	-Input (for classification) is a 2-D binary pattern -computationally efficient; -storage requirements reasonable; -especially suited for (optical) character recognition.
15	Yes	Min. total cost (feature meas. cost + classif. cost).	-Have design samples; -conditional densities are unimodal.	-Provides a binary tree; -superior performance to the method proposed in ref. #13; both in terms of simplicity of design and the total expected cost of classification.
16	No	"	–	-Heuristic; - provides a binary tree.
46	Yes	Weighted sum of P_e and avg. meas. cost in classifying a random sample	-Uses ML at each node -overall P_e of tree is small; -features on any path from root to a terminal node are statist. indep	-A bottom-up approach; -time and space complexities are high.
53	No	–	-Training sample avail.	-Unsupervised clustering of a priori classes.
54	No	Accuracy	-Labeled training pattern available.	-Examines 1 feature at each node; -finds best feature and corresp. threshold for each node.
58	Yes	A general cost function.	Given: samples from all the classes.	-Binary tree; -storage requirements may be large.
59	Yes	"	"	-Single feature comparison at each node; -feasible for small # of features; -given a specific partitioning of the feature space (i.e., P_e fixed), this method could be used to obtain a tree (for instance) with min. # of nodes (i.e., computation).
66	Locally	Min. Bayes error.	-Two-class problem; -training samples available.	-Integrates binary tree construction and feature selection problem; -efficient for 2-class problem; -can evaluate contribution of each feature to the classification task.
69	Yes	Prob. of error	-All the N features are avail. at every node.	-Estimates a posteriori prob.'s, but in a sequence of steps; -asymptotically robust w.r.t. changes in tree structure.
72	Locally	Max. mutual inf. gain at each node.	Given: a set of design samples & a pre-spec. desired P_e.	-Examines 1 feature per node; - provides a binary structure; -easy to implement; -feature selection capability inherent in the algorithm.
74	No	Min. expected cost of a tree; weights of the terminal nodes plus meas. cost	-Meas. costs are known -each meas. can be used only once in decision making.	-Useful for discrete feature applications; -efficient; -easy to implement.
77	Yes	Max. entropy reduction and min. overlap.	–	-A clustering algorithm; -useful for large # of classes

Table 1- Summary of tree structure design - continued.

78	sub-opt.	weighted sum of accuracy &efficiency.	-Decision rule at each stage is Gaussian ML	-Many applications in remote sensing explained here.
95	Yes	-Accuracy, or -total perform.	-Uses training set to estimate appropriate densities.	–
97	No	Efficiency and accuracy.	-Multivariate Gaussian dist. for all the classes	Uses linear discriminant function to improve speed; -number of features at each node is pre-specified.

Table 2- Summary of feature selection

Reference	Optimal	Criterion?	Remarks (by author or alternate reference)
46	Yes	Accuracy	Not very efficient.
54	No	Accuracy	Finds one best feature at each node
66	Locally	Accuracy	" " "
69	Yes	Max. accuracy in estimating proper a posteriori prob.'s	—
72	No	Max. inf. gain at each node.	Finds one best feature at each node.
[53],[78],[95], [97]	No	Max. separability between classes at each node, usually measured by some distance function.	Usually by some heuristic methods number of features to be used at each node are specified; most practical (feasible) ones.

Table 3- Summary of decision rule & search strategies

Reference	Optimal?	Criterion?	Remarks
41	No	Min. total cost of decision making;i.e.,sum of costs at different nodes plus risk of final classification.	-Several search methods within the context of pattern classification are addressed here.
46	Yes	Weighted sum of correct recognition rate and avg.meas. cost incurred per sample.	-Assumes knowledge of all joint class conditional probabilities; -some methods to reduce computational complexities are offered.
48	No	–	-Crucial task is to find tight bounds on the goal risk; -able to back-up & re-route, thus can improve correct recogn. rate.
[50],[51]	Yes	Probability of error	-Assumes: tree structure and feature subsets to be used at each node are given.
81	No	Allow trade-off between accuracy & efficiency.	-Uses: fuzzy logic search to improve speed, and global training to improve correct recogn. rate.

Table 4- Cited applications

Reference	Cited Application
[4],[6],[42], [78],[95],[97]	Remote sensing
[53],[59],[98]	Medical
[15],[16]	Speech
[8],[9],[10],[30],[54] [69],[72],[74] [81]-[84]	Character recognition
[7],[41],[46]-[49] [58],[62]	General

REFERENCES

[1] J. Aczel and J. Daroczy, *On measures of information and their characterizations*, New York : Academic, 1975.

[2] Rudolf Ahlsmede and Ingo Wegeru, *Search problems*, Wiley-Interscience, 1987.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullmann, "The Design and Analysis of Computer Algorithm," Reading, MA, Addison- Wesley, 1974.

[4] P. Argentiero, R. Chin and P. Beaudet, "An automated approach to the design of decision tree classifiers," IEEE Trans. Pattern Anal. Mach. Intell. PAMI-4, 51-57 (1982).

[5] L. Atlas, et. al. , " Performance comparison between backpropagation networks and classification trees on three real-world applications," ISDL report. Also to appear in NIPS proceedings publications 1990.

[6] L. A. Bartolucci, P. H. Swain, and C. Wu, " Selective radiant temperature mapping using a layered classifier," IEEE trans. Geosci. Electron. vol. GE-14, 101-106 (1976).

[7] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and regression trees*, Belmont, CA: Wadsworth Int. 1984.

[8] R. C. Casey and G. Nagy, "Decision tree design using a probabilistic model," IEEE Trans. Inform. Theory 30, 93-99 (1984).

[9] R. L. P. Chang, "Application of fuzzy decision techniques to pattern recognition and curve fitting," Ph.D. Thesis, Dept. of EECS, Princeton Univ., 1976.

[10] R. L. P. Chang and T. Pavlidis, "Fuzzy decision tree algorithms," IEEE Trans Syst. Man Cybernet., SMC-7, 28-35 (1977).

[11] P. A. Chou and R. M. Gray, "On decision trees for pattern recognition," IEEE Symposium on Information Theory, Ann Arbor MI., 69, 1986.

[12] M. Chou, T. Lookabaugh, and R. M. Gray, " Optimal pruning with applications to tree structured source coding and modeling," IEEE Trans. Inform. Theory, vol. IT-35, 299-315 (1989).

[13] G. R. Dattatreya and L. N. Kanal, "Adaptive pattern recognition with random costs and its application to decision trees," IEEE Trans. Syst. Man Cybernet., SMC-16, No.2, 208-218 (1986).

[14] G. R. Dattatreya and L. N. Kanal, " Decision trees in pattern recognition," In *Progress in Pattern Recognition 2*, Kanal and Rosenfeld (eds.) , Elsevier Science Publisher B.V., 189-239 (1985).

[15] G. R. Dattatreya and V. V. S. Sarma, "Decision tree design for pattern recognition including feature measurement cost," in Proc. 5th Int. Conf. Pattern Recognition, vol. II, 1212-1214 (1980).

[16] G. R. Dattatreya and V. V. S. Sarma,"Bayesian and decision tree approaches for pattern recognition including feature measurement costs," IEEE Trans. Pattern Anal. Mach. Intell. PAMI-3, 293-298, (1981).

[17] E. Diday and J. V. Moreau, "Learning hierarchical clustering from examples -- Applications to the adaptive construction of dissimilarity indices," Pattern Recognition Lett.," 223-230 (1986)

[18] R.O. Duda and P.E. Hart, *Pattern classification and scene analysis*, Wiley-Interscience, 1973.

[19] P. Fletcher and M.j.D. Powell,"A rapid decent method for minimization," Computer Journal, Vol.6, ISS.2, 163-168 (1963).

[20] J. H. Friedman, "A variable metric decision rule for nonparametric classification," Stanford Linear Accelerator Center-PUB-1573, CS-75-487, Apr. 1975.

[21] ----------, " A recursive partitioning decision rule for nonparametric classifier, IEEE Trans. Comput., vol. C-26, 404-408 (1977).

[22] K.S. Fu, *Sequential methods in pattern recognition and machine learning*, Academic press, 1968.

[23] K. Fukunaga, *Introduction to statistical pattern recognition*, Academic Press: New York, 1972.

[24] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-Nearest Neighbors," IEEE Trans. Comput. C-24, 750-753 (1975).

[25] S.B. Gelfand, C.S. Ravishankar, and E.J. Delp, "An iterative growing and pruning algorithm for classification tree design," to appear in Proc. of Int. conference on Systems, Man, Cyber., Boston Massachusetts Nov. 1989.

[26] B.V. Gnedenko and V.S. Koreljuk,"On the maximum discrepancy between two empirical distributions," Dokl. Acad. Nauk., SSSR 4, 525-528 (1951).

[27] M. Golea and M. Marchand, " A growth algorithm for neural networks," Europhys. Lett., **12** (3), 205-210 (1990).

[28] S. W. Golomb and L. D. Bavmert, "Backtrack programming," J. Ass. Comput. Mach., vol. 12, 516-524 (1965).

[29] R. M. Goodman and P. Smyth, "Decision tree design from a communication theory standpoint," IEEE Trans. Inform. Theory IT-34, 979-994 (1988).

[30] Y.X. Gu, Q.R. Wang and C.Y. Suen,"Application of multi-layer decision tree in a computer recognition of Chinese characters," IEEE Trans. Pattern Anal. Machine Intell., Vol. PAMI-5, 83-89 (1983).

[31] S. Gelfand and H. Guo, "Tree structured classifiers with multilayer neural network decision nodes," work in progress at Purdue University.

[32] H. Guo and S.B. Gelfand, "Convergence analysis of Back propagation algorithm for feed forward neural networks," submitted to IEEE Trans. on Systems and circuits.

[33] D. E. Gustafson, S. B. Gelfand, and S. K. Mitter, " A nonparametric multiclass partitioning methods for classification," in proc. 5th int. conf. pattern Recognition, 654-659 (1980).

[34] P. A. V. Hall, "Branch-and-Bound and Beyond," Proc. 2nd Joint Int. Conf on Artificial Intelligence, 1971.

[35] R.M. Haralick,"The table look-up rule," in Proc. Conf. on Pattern Recognition, 447- ,1976.

[36] P. E. Hart, "Searching probabilistic decision tress," AI Group Tech. Note No. 2, SRI project 7494, SIR, Stanford, CA., 1969.

[37] C. R. P. Hartmann, P. K. Varshney, K. G. Mehrotra, and C. L. Gerberich," Application of information theory to the construction of efficient decision trees," IEEE Trans. Inform. Theory IT-28, No.4 565-577 (1982).

[38] E. G. Henrichon,Jr. and K. S. Fu, "A nonparametric partitioning procedure for pattern classification," IEEE Trans. Comput., Vol. C-18, 604-624, (1969).

[39] G. F. Hughes, "On the mean accuracy of statistical pattern recognizers," IEEE Trans. Inform. Theory IT-14, 55-63 (1968).

[40] L. Hyafil and R. L. Rivest, " Constructing optimal binary decision trees is NP-complete ," Information Processing Letters, Vol. 5, No. 1, 15-17 (1976).

[41] L. N. Kanal, "Problem-solving methods and search strategies for pattern recognition," IEEE Trans. Pattern Anal. Mach. Intell. PAMI -1, 193-201 (1979).

[42] B. Kim and D. A. Landgrebe, " Hierarchical decision tree classifiers in high-dimensional and large class data," Ph.D. Thesis and Technical Report TR-EE-90-47, School of EE, Purdue University (1990).

[43] P. R. Krishnaiah, Ed. "On hierarchical classifier and interactive design, in Applications of Statistics," Amsterdam, The Netherlands: North-Holland, 1971, pp 301-321.

[44] D. E. Knuth, "Optimum binary search trees," ACTA Informatica, vol. 1, 14-25 (1971).

[45] D. E. Knuth, "The art of computer programming,1: fundamental algorithms," Addison-Wesley 1968.

[46] A. V. Kulkarni and L. N. Kanal, "An optimization approach to hierarchical classifier design," Proc. 3rd Int. Joint Conf. on Pattern Recognition, San Diego, CA, 1976.

[47] A. V. Kulkarni and L. N. Kanal, "Admissible search strategies for parametric and non-parametric hierarchical classifiers," Proc. 4th Int. Conf. on Pattern Recognition, Kyoto, Japan, 1978.

[48] A. V. Kulkarni, "On the mean accuracy of hierarchical classifiers," IEEE Trans. Comput. C-27, 771-776 (1978).

[49] A. V. Kulkarni, "Optimal and heuristic synthesis of hierarchical classifiers," Ph.D. dissertation, Univ. of Maryland, College Park, Comput. Sci. Tech. Rep. TR-469, 1976.

[50] M. W. Kurzynski, "Decision rules for a hierarchical classifier," Pattern Recognition Lett. 1, 305-310, (1983).

[51] M. W. Kurzynski, "The optimal strategy of a tree classifier," Pattern Recognition 16, 81-87 (1983).

[52] E. L. Lawler and D. E. Wood, "Branch-and-Bound: A Survey," Operation Research, vol. 14, 1966.

[53] G. Landeweerd, T. Timmers, E. Gelsema, M. Bins and M. Halic, "Binary tree versus single level tree classification of while blood cells," Pattern Recognition 16, 571-577 (1983).

[54] X. Li and R. C. Dubes, "Tree classifier design with a Permutation statistic," Pattern Recognition 19, 229-235 (1986).

[55] Y. K. Lin and K. S. Fu, " Automatic classification of cervical cell using a binary tree classifier," Pattern Recognition 16, 69-80 (1983).

[56] R. P. Lippmann, "An introduction to computing with neural nets," IEEE ASSP magazine, 4-22, April (1987).

[57] C. McMillan Jr.,*Mathematical programing*, Wiley, Chapt.8 (1975).

[58] W. S. Meisel and D. A. Michalopoulos,"A Partitioning algorithm with application in pattern classification and optimization of decision trees," IEEE Trans. Computers, Vol. C-22, 93-103, Jan. 1973.

[59] J. Mui and K. S. Fu, "Automated classification of nucleated blood cells using a binary tree classifier," IEEE Trans. Pattern Anal. Mach. Intell. PAMI-2, 429-443 (1980).

[60] M. Nadler, "Error and reject rates in a hierarchical pattern recognizer," IEEE Trans. Comput. C-19, 1598-1601 (1970)

[61] N.J. Nilsson, *Problem-solving methods in Artificial Intelligence*, New York : McGraw-Hill, 1971.

[62] H. Payne and W. Meisel, "An algorithm for constructing optimal binary decision trees," IEEE Trans. Computing C-26, 905- 916 (1977).

[63] J. R. Quinlan, " Induction of decision trees, " Machine Learning 1, 81- 106 (1986).

[64] ----------, " Decision tress and decsion-making, " IEEE trans. on Systems, Man, Cyber. vol. SMC-20, No. 2, 339-346, (1990).

[65] J. R. Quinlan and R. L. Rivest, " Inferring decision trees using the minimum description length principle," Information and Computation, vol. 80, no. 3, 227-248 (1989).

[66] E. Rounds, "A combined non-parametric approach to feature selection and binary decision tree design," Pattern Recognition 12, 313-317 (1980).

[67] E. Rounds, "Computation of two-sample Kolmogorov-Smirnov statistic. Project Memo TSC-PM-A 142-5, Technology Service Corporation, Santa Monica, CA., 1978.

[68] D.E. Rumelhart and J.L. McClelland (eds), "Parallel distributed processing," vol. 1 M.I.T. press, Cambridge Massachusetts, 1986.

[69] J. Schuermann and W. Doster, "A decision-theoretic approach to hierarchical classifier design," Pattern Recognition 17, 359- 369 (1984).

[70] I. K. Sethi, " Entropy nets: From decision trees to Neural networks," to appear in proc. IEEE special issue on Neural Networks September 1990.

[71] ----------, " Layered neural net design through decision trees," in proc. Int. Symposium on Circuits and Systems, New Orlean, LA. May 1-3 1990.

[72] I.K. Sethi and B. Chatterjee,"Efficient decision tree design for discrete variable pattern recognition problems," Pattern Recognition 19,197-206 (1977).

[73] I. K. Sethi and M. Otten, " Comparison between entropy net and decision tree classifiers, " in proc. Int. Joint Conf. on Neural Net. IJCNN, San Diego, CA 1989.

[74] I. K. Sethi and G. Sarvarayudu, "Hierarchical classifier design using mutual information," IEEE Trans. Pattern Anal. Mach. Intell. PAMI-4, 441-445 (1982).

[75] Q. Y. Shi," A method for the design of binary tree classifiers, " in proc. IEEE conf. Image processing and Pattern Recognition, 21-26 (1981).

[76] C. R. Smith and W. T. Grandy, Jr. eds., *Maximum-Entropy and Bayesian Methods in Inverse Problems,"* Reidel, Holland, 1985.

[77] C. Y. Suen and Q. R. Wang, "ISOETRP - An interactive clustering algorithm with new objectives," Pattern Recognition 17, 211-219 (1984).

[78] P. Swain and H. Hauska, "The decision tree classifier design and potential," IEEE Trans. Geosci. Electron, GE-15, 142-147 (1977).

[79] P. E. Utgoff, " Incremental induction of decision tress," Machine Learning 4, 161-186 (1989).

[80] A. Wald, *Sequential analysis*, Wiley, New York, 1947

[81] Q. R. Wang and C. Y. Suen, "Large tree classifier with heuristic search and global training," IEEE Trans. Pattern Anal. Machine Intell., PAMI-9, 91-102 (1987).

[82] Q. R. Wang and C. Y. Suen, "Analysis and design of a decision tree based on entropy reduction and its application to Large character set recognition," IEEE Trans. Pattern Anal. Machine Intell., PAMI- 6, 406-417 (1984).

[83] Q. R. Wang and C. Y. Suen, "Classification of Chinese characters by phase feature and fuzzy logic search," Proc. Int. Conf. Chinese Inform. Processing, vol. 1, Beijing, 133-155 (1983).

[84] Q. R. Wang, Y. X. Gu, and C. Y. Suen, "A Preliminary study on computer recognition of Chinese characters printed in different fonts," Proc. Int. Conf. Chinese Lang. Comput. Soc., 344-351 (1982).

[85] Watanabe, "Pattern recognition as conceptual morphogenesis," IEEE Trans. Pattern Anal. Machine Intell. 2, 161-165 (1980).

[86] S. Watanabe, *Knowing and guessing*, Wiley, New York (1969).

[87] S. Watanabe, *Pattern recognition: human and machine*, Wiley-Interscience, 1985.

[88] S. Watanabe, "Learning process and inverse H-theorem," IRE Trans. Inform. Theory IT-8, 246 (1962).

[89] S. Watanabe, "Information-theoretical aspects of inductive and deductive inference," IBM J. Res. Develop., vol. 4, 208. 1960.

[90] S. Watanabe, "Application of dynamical coalescence model of clustering," in Proc. 3rd Int. Joint Conf. on Pattern Recognition, San Diego, 176, 1976.

[91] S. Watanabe, "Karhunen-Loeve expansion and factor analysis," in Trans. 4th Prague Conf. on Inform. Theory, etc., 1965.

[92] S. Watanabe and E. T. Harada, "A dynamical model of clustering," in *Proc. 2nd. Int. Joint Conf. on Pattern Recognition*, Copenhagen, 413, 1974.

[93] S. Watanabe, "Pattern Recognition as a quest for minimum entropy," Pattern Recognition 13, 381-387, 1981.

[94] R. Winston, "A heuristic program that constructs decision trees," MIT project MAC, Memo #173, 1969.

[95] C. Wu, D. Landgrebe, and P. Swain, The decision tree approach to classification," School Elec. Eng., Purdue Univ., Lafayette, IN, Rep. RE-EE 75-17, 1975.

[96] S. S. Yau and J. M. Garnet, "Least mean square approach to pattern classification," Frontiers of Pattern Recognition, 575-588, Academic Press: New York, 1972.

[97] K. C. You and K. S. Fu, "An approach to the design of a linear binary tree classifier," *Proc. 3rd Symp. Machine Processing of Remotely Sensed Data*, Purdue Univ. 1976.

[98] S. Q. Yun and K. S. Fu, "A method for the design of binary tree classifiers," *Pattern Recognition* 16, 593-603 (1983).

[99] L. A. Zadeh, "Fuzzy sets," *Inform. Control*, 338-353, 1965.