Optimal Partitioning for Classification and Regression Trees

Philip A. Chou, *Member, IEEE*

***Abstract*—In designing a decision tree for classification or regression, one selects at each node a feature to be tested, and partitions the range of that feature into a small number of bins, each bin corresponding to a child of the node. When the feature's range is discrete with *N* unordered outcomes, the optimal partition, that is, the partition minimizing an expected loss, is usually found by an exhaustive search through all possible partitions. Since the number of possible partitions grows exponentially in *N*, this approach is impractical when *N* is larger than about 10 or 20. In this paper, we present an iterative algorithm that finds a locally optimal partition for an arbitrary loss function, in time linear in *N* for each iteration. The algorithm is a *K*-means like clustering algorithm that uses as its distance measure a generalization of Kullback's information divergence. Moreover, we prove that the globally optimal partition must satisfy a nearest neighbor condition using divergence as the distance measure. These results generalize similar results of Breiman *et al.* to an arbitrary number of classes or regression variables and to an arbitrary number of bins. We also provide experimental results on a text-to-speech example, and we suggest additional applications of the algorithm, including the design of variable combinations, surrogate splits, composite nodes, and decision graphs.**

***Index Terms*—Clustering, decision trees, information divergence, text-to-speech.**

I. Introduction

A CLASSIFICATION or regression tree is a binary tree, not necessarily balanced, that given an input X produces an output $\hat{Y}$ that approximates some random variable of interest Y, stochastically related to X. This deterministic mapping is accomplished as follows. Associated with each internal node of the tree is a binary function of the input X, and associated with each external node is a specific output label $\hat{Y}$. Starting at the root node, the binary function is used to test the given input X. If the result is "0", the left branch is followed; if the result is "1", the right branch is followed. The process is repeated until reaching an external node, or leaf, at which point the associated label $\hat{Y}$ is output. The tree is designed to minimize (at least approximately) the expected loss between Y and $\hat{Y}$.

As an example, consider the classification tree of Fig. 1 for an optical character recognition (OCR) problem. With Y a letter in {"a",$\cdots$,"z"}, and X a feature vector $(X_1, \cdots, X_8)$ whose components, shown in Table I, are derived from a noisy image of Y, this tree attempts to classify X by testing one component at each node. The root node, for example, tests component X_8. If $X_8 \in \{1, 3, 7\}$, then the left branch from the root is followed, otherwise the right branch is followed. The image of a character with feature vector $(1,1,1,1,1,1,2,7)$ would be mapped into $\hat{Y}$ = "i", based on the fact that $X_8 = 7$ = "two or more components" and $X_6 = 1$ = "no southeast concavity." Note that many different feature vectors may map to the same leaf, and that many different leaves may have the same label. Furthermore, some classes may not be represented by any label. The tree is designed so that the probability of error, or the expected loss between Y and $\hat{Y}$, is low, where the loss here is the misclassification cost: 1 if $Y \neq \hat{Y}$ and 0 otherwise.

The difference between a classification tree and a regression tree is that in a classification tree, Y is "categorical" (i.e., takes values in a discrete set), whereas in a regression tree, Y is "continuous" (i.e., real-valued) and can be either a scalar or vector. Classification tree performance is usually given in terms of probability of error; regression tree performance is usually given in terms of mean squared error. In general, performance is measured by expected loss, for some appropriate loss function. Classification trees, also called decision trees in the literature, have been well-studied, with applications including pattern recog-

TABLE I
FEATURES FOR OCR EXAMPLE

Feature	Possible Outcomes	
X_1 (north concavities)	1	no concavity
	2	shallow concavity
	3	deep concavity
	4	two concavities
	5	three or more concavities
X_2 (south concavities)	1	no concavity
	2	shallow concavity
	3	deep concavity
	4	two concavities
	5	three or more concavities
X_3 (northwest concavities)	1	no concavity
	2	northwest concavity
X_4 (northeast concavities)	1	no concavity
	2	northeast concavity
X_5 (southwest concavities)	1	no concavity
	2	southwest concavity
X_6 (southeast concavities)	1	no concavity
	2	southeast concavity
X_7 (vertical bars)	1	no vertical bars
	2	one narrow vertical bar
	3	two vertical bars
	4	three vertical bars
	5	four vertical bars
	6	five or more vertical bars
	7	one wide bar on the right
	8	one wide bar on the left
X_8 (horizontal lines and loops)	1	simple line
	2	complicated line
	3	simple loop
	4	complicated loop
	5	exactly two loops
	6	three or more loops
	7	two or more components

Manuscript received September 28, 1989; revised May 16, 1990. This work was supported in part by the National Science Foundation under Grant IST-8509860.

The author is with the Xerox Palo Alto Research Center, Palo Alto, CA 94304. This work was performed while he was with the Department of Electrical Engineering, Stanford University, and with the Department of Signal Processing Research, AT&T Bell Laboratories.

IEEE Log Number 9041537.

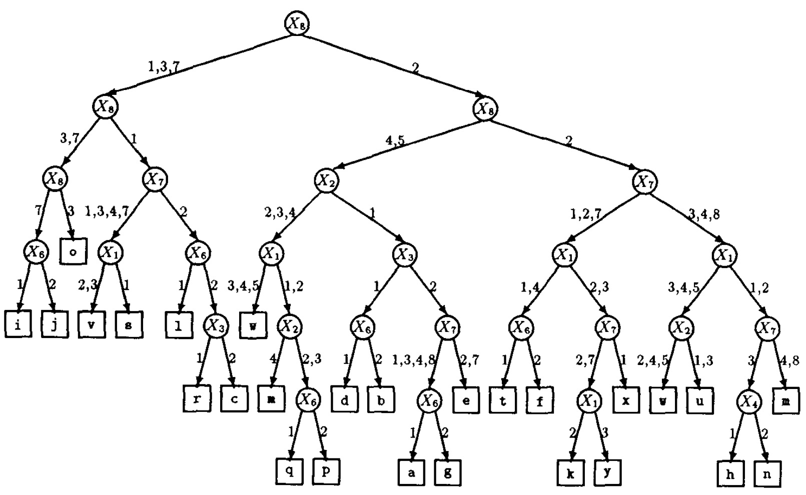

Fig. 1. Classification tree for OCR example.

nition [1]–[10], logic design [11], taxonomy, questionnaires, and diagnostic manuals [12], expert systems, and machine learning [13]–[20], and the conversion of decision tables to nested "if ... then ... else" rules for computer programs [21]–[31]. Regression trees have also found applications in a number of areas, including least squares regression [32], [33], [7] and vector quantization [34]–[36].

Due to the inherent computational complexity of constructing optimal trees (i.e., trees having the minimal expected loss for their size) [37], [38], practical procedures for constructing trees are almost universally steepest-descent greedy procedures that "grow" trees outward from the root. Each step of such a procedure operates on a partially grown tree by splitting some terminal node into two children, making it a parent node. The parent node is assigned a binary *test*, or function of the input $\boldsymbol{X}$, that among some collection of tests permitted at that node, most improves the performance of the new tree. Thus the procedure is stepwise optimal.

In a straightforward version of the growing procedure, an exhaustive search through the collection of permissible tests is performed at each node. If the collection of tests is large, then the run time of the growing procedure is also large. In particular, if the feature vector $\boldsymbol{X} = (X_1, \cdots, X_J)$ includes a categorical feature variable X taking values in a finite set $A = \{x_1, \cdots, x_N\}$, say, then the collection of permissible tests on X includes the tests

$$\alpha(X) = \begin{cases} 0 & \text{if } X \in A_0 \\ 1 & \text{if } X \in A_1 \end{cases}$$

for each partition A_0, A_1 of A ($A_0 \cup A_1 = A$ and $A_0 \cap A_1 = \emptyset$). Since the number of such partitions is 2^N, the run time of the growing procedure is exponential in the size of the alphabet N.

For small problems, such as the OCR problem in which the feature with the largest alphabet has only $N = 8$ possible outcomes, this exponential run time may not present much difficulty. But for larger problems, e.g., an OCR problem in which the feature vector also includes the class assigned to the previous character, the number of permissible tests at every node becomes more than 2^{26}, and the run time of the growing procedure becomes impossibly large. Much larger collections of tests are just as easy to imagine.

For some problems, algorithms for finding optimal partitions in *linear time* (in N) have been discovered. In 1958, for example, W.D. Fisher noticed that when Y is real-valued (the scalar regression case), the least squares partition A_0, A_1 of A is contiguous, in the sense that

$$E[Y \mid X = x] \leq E[Y \mid X = \bar{x}]$$

for all $x \in A_0$ and $\bar{x} \in A_1$ [39]. Thus to find the optimal least squares partition it suffices to consider only the $N - 1$ contiguous partitions, rather than all 2^N partitions.

In 1984, Breiman *et al.* extended Fisher's result to the case when Y is binary (the two-class case), and to arbitrary convex $\cap$ impurity measures [7, Theorem 4.5]. (The relationship between impurity measures and loss functions is described in Section II.) These results can be viewed geometrically as follows. If the N points $E[Y \mid X = x]$ for $x \in A = \{x_1, \cdots, x_N\}$ are plotted on the real line, then there is a threshold such that all the x's corresponding to points below the threshold belong to the optimal A_0 and all the x's corresponding to points above the threshold belong to the optimal A_1. Therefore it suffices to evaluate each of the $N - 1$ possible thresholds, and choose the partition with the best performance.

In 1988, Chou extended Breiman's result to arbitrary numbers of classes (in the classification case) under the log-likelihood loss function, and to vectors of arbitrary length (in the regression case) under the squared error loss function [10]. Briefly, the threshold of Breiman *et al.* was generalized to a hyperplane for these cases, and the set of possibly optimal partitions was searched in linear time per iteration by an iterative descent algorithm formally equivalent to the K-means algorithm [40]–[44] or the generalized Lloyd algorithm [45], [46], but using a different distance measure depending on the loss function. The results were also extended to locally optimal K-ary partitions, $K \geq 2$. This led to trees of degree $K \geq 2$, and more usefully, to directed acyclic decision graphs, called decision trellises.

The present paper is the journal version of the optimal partitioning results of [10]. In addition, the present paper generalizes the results of [10] to arbitrary loss functions, in a unified mathematical framework, and describes a number of other applications of optimal partitioning within the context of decision tree design. The Appendix catalogues some fairly general loss functions.

Independently, Burshtein *et al.* generalized the results of [10] in another direction: to arbitrary convex $\cap$ impurity measures, and proposed a polynomial time algorithm based on linear programming for finding the globally optimal binary partition [47], [48]. Unfortunately their algorithm is exponential in the number of classes (in the classification case) or the length of the vector (in the regression case). Furthermore it is valid only for the binary ($K = 2$) case. It is, however, guaranteed to find the globally optimal partition, whereas the iterative descent algorithm is not.

The present paper unfolds as follows. The next section, Section II, develops the relationship between the loss functions used to measure tree performance and the measures of node impurity used in [7], [47], [48], and introduces the notion of divergence, the "distance" to be used in the iterative descent algorithm. Section III presents the main theorem—necessary conditions on the form of the optimal partition in higher dimensions—and proves it constructively. This construction leads directly to the iterative descent algorithm, which is also presented in Section III. Section IV applies the iterative descent algorithm to splitting nodes within the greedy growing algorithm during the design of a decision tree for a text-to-speech system. Section V suggests further applications of the algorithm in decision tree design, including variable combinations, surrogate splits, composite nodes, higher order splits, and decision trellises, Section VI is a discussion and conclusion.

II. Loss, Impurity, and Divergence

We begin with the *loss function* $\ell(y, \hat{y})$, which measures the loss, or cost, incurred by representing the object y by the approximation $\hat{y}$. Typical examples of loss functions are the misclassification error

$$\ell(y,\hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases},$$

used in classification, and the squared error

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2,$$

used in regression. Weighted versions of these also exist. (See Appendix A.) We do not restrict $\hat{y}$ to have the same alphabet as y. For example, in the M-class case, if y takes values in $\{1, \cdots, M\}$ and $\hat{\boldsymbol{y}}$ is any probability vector $\hat{\boldsymbol{y}} = (\hat{p}(1), \cdots, \hat{p}(M))$, then the log likelihood loss function is defined

$$\ell(y, \hat{\boldsymbol{y}}) = -\log \hat{p}(y).$$

(This can be interpreted as the number of bits required to specify y when using an entropy code matched to $\hat{\boldsymbol{y}}$, and hence is useful in designing decision trees for data compression [49].) However, we will usually take both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ to be M-dimensional vectors (hence the boldface notation). In the case of classification with M classes, $\boldsymbol{y}$ will be the *class indicator vector* whose components are all "0", except for the yth component, which is "1". We will use the nonbold symbol y when necessary to represent the index of the class in $\{1, \cdots, M\}$.

The particular loss function chosen for a given application may be motivated by any number of things: physical (e.g., perceptual) criteria, theoretical properties, standard convention, or a combination of these. Selection of the loss function is not addressed in this paper. However, a number of special loss functions are treated in Appendix A.

Since a classification or regression tree represents a random object $\boldsymbol{Y}$ by a deterministic mapping $\hat{\boldsymbol{Y}} = q(\boldsymbol{X})$, say, we can measure the performance of the tree by the expected loss, or risk,

$$R(q) = E[\ell(\boldsymbol{Y}, q(\boldsymbol{X}))]. \tag{1}$$

Here, of course, we are assuming $\boldsymbol{X}$ and $\boldsymbol{Y}$ are jointly distributed random objects on an underlying probability space. In practice, the risk (1) is evaluated by taking sample averages. Hence *validation,* or the process of verifying the risk on independent data, is an important aspect of tree design. However, validation is not our primary concern here. In this paper, we simply design with a training sequence of $(\boldsymbol{X}, \boldsymbol{Y})$ pairs and validate with a separate test sequence. (The more sophisticated method of cross-validation [7] could also be used.) All probabilities and expectations in this paper may be respectively interpreted as sample distributions and sample averages of a training sequence.

We can express the risk (1) as a nested expectation by conditioning on the leaves of the tree, as follows. Let T denote the set of nodes in the tree, and take each $t \in T$ to be an event in the original probability space. Thus $P(t)$ is the probability that node t is reached when $\boldsymbol{X}$ is classified. Let $\tilde{T} \subseteq T$ denote the subset of leaves of T. We can see that the set of leaves $\tilde{T}$ forms a partition of the sample space. Hence the risk (1) can be rewritten

$$R(q) = \sum_{t \in \tilde{T}} P(t) \cdot E[\ell(\boldsymbol{Y}, \hat{\boldsymbol{y}}(t)) \mid t], \tag{2}$$

where $\hat{\boldsymbol{y}}(t)$ is the output label at leaf t.

At each node t, the constant output label $\hat{\boldsymbol{y}}(t)$ that minimizes the conditional expected loss $E[\ell(\boldsymbol{Y}, \hat{\boldsymbol{y}}(t)) \mid t]$ will be called the *centroid* of t, which we shall denote

$$\mu(t) = \arg\min_{\boldsymbol{y}} E[\ell(\boldsymbol{Y}, \hat{\boldsymbol{y}}) \mid t].$$

(Here and throughout the paper, $\arg\min_x f(x)$ denotes any x, not necessarily unique, that minimizes f.) The minimum value of this expected loss,

$$i(t) = E[\ell(\boldsymbol{Y}, \mu(t)) \mid t],$$

will be called the *impurity* of t. With these definitions, it is clear from (2) that

$$R(q) \geq \sum_{t \in \tilde{T}} P(t) i(t). \tag{3}$$

If the output labels are chosen to be the centroids of their nodes, then (3) hold with equality, so that

$$R(q) = \sum_{t \in \tilde{T}} P(t) i(t). \tag{4}$$

This is assumed in the sequel.

Impurity has the following convexity property. Let the node t be split into left and right children t_0 and t_1. (This corresponds to splitting the event t into two events t_0 and t_1.) Then by the definitions of $i(t)$ and $\mu(t)$,

$$\begin{aligned} i(t) &= P(t_0 \mid t) E[\ell(\boldsymbol{Y}, \mu(t)) \mid t_0] + P(t_1 \mid t) E[\ell(\boldsymbol{Y}, \mu(t)) \mid t_1] \\ &\geq P(t_0 \mid t) i(t_0) + P(t_1 \mid t) i(t_1). \end{aligned} \tag{5}$$

That is, the average impurity of a node never increases when the node splits. Multiplying (5) by $P(t)$, we obtain

$$P(t)i(t) \geq P(t_0)i(t_0) + P(t_1)i(t_1).$$

Hence we see that the overall risk (4) of a tree likewise never increases when a node splits. Moreover, when a node is split, the decrease in the tree's overall risk is equal to the decrease in the node's average impurity (times the probability of the node).

In the greedy growing procedure, at each node we seek the split, or binary test of $\boldsymbol{X}$, that most reduces the overall risk of the tree, i.e., most improves its performance. But by what we have just said, this is equivalent to finding the split that most reduces the average impurity of the node.

Breiman *et al.* [7] and Burshtein *et al.* [47], [48] use a slightly different definition of impurity. They *define* $\mu(t)$ to be the expected value of $\boldsymbol{Y}$ given t, $E[\boldsymbol{Y} \mid t]$, and they let ϕ be an arbitrary convex $\cap$ functional. Then they define the impurity to be $i(t) = \phi(\mu(t))$. This will also have the convexity property (5), owing to the convexity of ϕ. As pointed out in [48], such a formulation can be used to minimize the risk, or expected loss, in those cases where the loss function can be expressed

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \ell_0(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \ell_1(\boldsymbol{y}), \tag{6}$$

where $\ell_0(\boldsymbol{y}, \hat{\boldsymbol{y}})$ is affine in $\boldsymbol{y}$. Note that the squared error $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_m (y_m - \hat{y}_m)^2$ satisfies (6), as does the misclassification error $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_m (1 - y_m)\hat{y}_m$, where $\boldsymbol{y}$ is a class indicator vector and $\hat{\boldsymbol{y}}$ is a class probability vector. (See also Appendix A.) However, neither the absolute error $\sum_m |y_m - \hat{y}_m|$ nor the maximum error $\max_m |y_m - \hat{y}_m|$ satisfy (6). In case the loss function does satisfy (6), however, the functional ϕ may be defined

$$\phi(\mu) = \min_{\hat{\boldsymbol{y}}} \ell_0(\mu, \hat{\boldsymbol{y}}).$$

Then, defining $\mu(t) = E[\boldsymbol{Y} \mid t]$ and $i(t) = \phi(\mu(t))$, the split that most reduces the average impurity of a node, also most reduces the overall risk. In our work, the special form (6) is not assumed.

Finally, we introduce the notion of divergence. This is the key to formulating the partitioning algorithm as an iterative descent: divergence is needed to play the role of the metric. Suppose an arbitrary output label $\hat{\boldsymbol{y}}$ is used in place of the centroid $\mu(t)$. The *divergence* of $\hat{\boldsymbol{y}}$ from t (or from $\mu(t)$) is defined to be the increase in expected loss when $\hat{\boldsymbol{y}}$ is used to represent $\boldsymbol{Y}$ instead of $\mu(t)$:

$$\begin{aligned} d(t, \hat{\boldsymbol{y}}) &= E[\ell(\boldsymbol{Y}, \hat{\boldsymbol{y}}) \mid t] - E[\ell(\boldsymbol{Y}, \mu(t)) \mid t] \\ &= E[\ell(\boldsymbol{Y}, \hat{\boldsymbol{y}}) \mid t] - i(t). \end{aligned}$$

Notice that by definition, $d(t, \hat{\boldsymbol{y}}) \geq 0$ for all $\hat{\boldsymbol{y}}$, with equality if $\hat{\boldsymbol{y}} = \mu(t)$ (although $\mu(t)$ is not necessarily unique).

This corresponds exactly to Kullback's *information divergence* [50], when the log likelihood loss function is used (hence our use of the name "divergence"). However, many other loss functions commonly used in classification and regression also induce divergences that are easily characterized. These include the weighted squared error, the minimum relative entropy, the Itakura–Saito distortion, the weighted misclassification error, and the weighted Gini criterion. These are catalogued in Appendix A.

III. Optimal Partitioning

The greedy growing algorithm for constructing classification and regression trees seeks at each node t, and for each categorical variable X in the feature vector $\boldsymbol{X} = (X_1, \cdots, X_J)$, the test or split or partition of X that most reduces the overall risk of the tree. We have seen that this is equivalent to seeking the partition that most reduces the average impurity of the node. Finding this optimal partition is the *partitioning problem.*

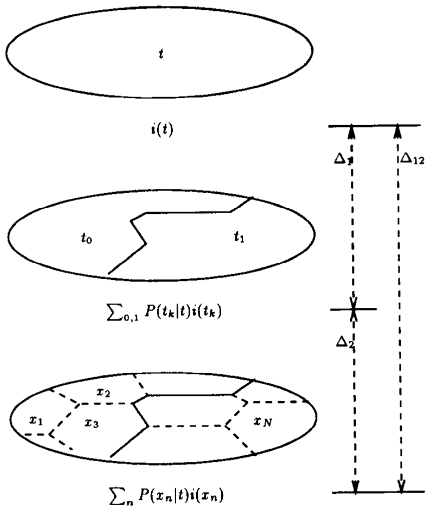

Fig. 2. Two-stage refinement of an event t.

More formally, let X be a discrete random variable with alphabet $A = \{x_1, \cdots, x_N\}$. Given an event t, the partitioning problem is to find a binary partition A_0, A_1 of A that minimizes the average impurity,

$$I(A_0, A_1 \mid t) = P(t_0 \mid t)i(t_0) + P(t_1 \mid t)i(t_1), \tag{7}$$

where the events $t_0 = t \cap \{X \in A_0\}$ and $t_1 = t \cap \{X \in A_1\}$ partition t according to whether X falls into A_0 or its complement A_1.

One way to view the problem is as a two-stage refinement of t, as shown in Fig. 2. We are given the coarsest partition of t, t itself, with impurity $i(t)$, and the finest partition of t, $\{x_1, \cdots, x_N\}$, with average impurity $\sum_n P(x_n \mid t)i(x_n)$. (Here, we are using the letters $x_1, \cdots, x_N$ to stand for the "atomic" events $t \cap \{X = x_n\}$, $n = 1, \cdots, N$.) We seek an intermediate partition of t, t_0, t_1, or equivalently A_0, A_1, with the least possible average impurity $\sum_{k=0,1} P(t_k \mid t)i(t_k)$.

The convexity property (5) assures us that the average impurity does not increase as the partition is refined. Hence the differences in average impurity, Δ_1, Δ_2, and Δ_{12}, shown in Fig. 2, must always be nonnegative. Indeed, they may be interpreted as average divergences:

$$\begin{aligned} \Delta_{12} &= i(t) - \sum_n P(x_n \mid t)i(x_n) \\ &= \sum_n P(x_n \mid t)\{E[\ell(\boldsymbol{Y}, \mu(t)) \mid x_n] - i(x_n)\} \\ &= \sum_n P(x_n \mid t)\, d(x_n, \mu(t)), \end{aligned} \tag{8}$$

$$\begin{aligned} \Delta_1 &= i(t) - \sum_{k=0,1} P(t_k \mid t)i(t_k) \\ &= \sum_{k=0,1} P(t_k \mid t)\{E[\ell(\boldsymbol{Y}, \mu(t)) \mid t_k] - i(t_k)\} \\ &= \sum_{k=0,1} P(t_k \mid t)\, d(t_k, \mu(t)), \end{aligned} \tag{9}$$

and

$$\Delta_2 = \sum_{k=0,1} P(t_k \mid t)i(t_k) - \sum_n P(x_n \mid t)i(x_n)$$

$$= \sum_{k=0,1} P(t_k \mid t)\left\{ i(t_k) - \sum_n P(x_n \mid t_k) i(x_n) \right\}$$

$$= \sum_{k=0,1} P(t_k \mid t) \sum_n P(x_n \mid t_k)\, d(x_n, \mu(t_k)). \tag{10}$$

Note that the sum $\Delta_{12} = \Delta_1 + \Delta_2$ is fixed, and that the average impurity (7) can be expressed

$$I(A_0, A_1 \mid t) = i(t) - \Delta_1$$

$$= \sum_n P(x_n \mid t) i(x_n) + \Delta_2. \tag{11}$$

Thus minimizing the average impurity of the intermediate partition is equivalent to either

1) maximizing the average divergence Δ_1, or
2) minimizing the average divergence Δ_2.

According to (9), maximizing Δ_1 corresponds to maximizing the weighted sum of divergences from t_0 and t_1 to the centroid $\mu(t)$ of t. Dually, according to (10), minimizing Δ_2 corresponds to minimizing the weighted sum of divergences from $x_1, \cdots, x_N$ to the centroids of their assigned bins, either $\mu(t_0)$ or $\mu(t_1)$.

To see the latter more clearly, let $\alpha : A \to \{0, 1\}$ be the function that assigns each letter in A to one of the two bins A_0 or A_1, and let $\beta(\cdot)$ be the function on $\{0, 1\}$ that assigns a centroid to each bin. That is, for each $x \in A$, let

$$\alpha(x) = \begin{cases} 0 & \text{if } x \in A_0 \\ 1 & \text{if } x \in A_1 \end{cases}$$

and for $k = 0, 1$, let

$$\beta(k) = \mu(t_k).$$

Then, since $P(x \mid t_k) = 0$ whenever $x \notin A_k$, the expression for Δ_2 (10) becomes

$$\Delta_2 = \sum_{k=0,1} P(t_k \mid t) \sum_{x \in A_k} P(x \mid t_k)\, d(x, \mu(t_k))$$

$$= \sum_{k=0,1} P(t_k \mid t) \sum_{x:\alpha(x)=k} P(x \mid t_k)\, d(x, \beta(\alpha(x))) \tag{12}$$

$$= \sum_x P(x \mid t)\, d(x, \beta(\alpha(x))). \tag{13}$$

Thus, Δ_2 is the weighted sum of the divergences from each x to the centroid of its assigned bin, either $\beta(0)$ or $\beta(1)$.

These weighted sums of divergences, Δ_{12}, Δ_1, and Δ_2, are illustrated in Figs. 3(a), (b), and (c), respectively. Events $t, t_0, t_1, x_1, \cdots, x_N$ are represented by their centroids, $\mu(t), \mu(t_0), \mu(t_1), \mu(x_1), \cdots, \mu(x_N)$, as points in a vector space. The divergences between them are shown as directed arcs. For example, $d(x_2, \mu(t))$ lies on the arc between $\mu(x_2)$ and $\mu(t)$. Imagine that each centroid has a mass equal to the probability of its associated event, e.g., the mass of $\mu(x_2)$ is $P(x_2 \mid t)$. Then, each divergence is weighted by the mass at its tail, and the weighted divergences are summed to obtain the average divergences Δ_{12} in Fig. 3(a), Δ_1 in Fig. 3(b), and Δ_2 in Fig. 3(c). The total weighted arc length in Fig. 3(a) is thus equal to the sum of the weighted arc lengths in Figs. 3(b) and (c).

Fig. 3(c) suggests that we can find the optimal partition A_0, A_1 of A by clustering $x_1, \cdots, x_N$ into two bins such that the weighted sum Δ_2 (13) is minimized. We can interpret the function α in (13) as assigning each x to one of the two clusters, and the function β as assigning a centroid to each of the two clusters.

The alternative interpretations of α and β are many. As we just said, we can interpret α as assigning each x_n to one of the

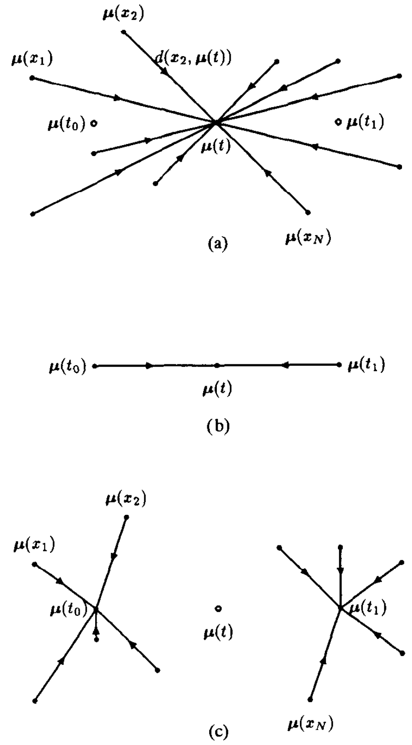

Fig. 3. Decomposition of average divergence.

two bins A_0, A_1. But we can also interpret α as assigning each outcome of X to the left or right child of a node, and β as assigning optimal output labels to the left and right children. Or, we can interpret α as the "split" at a node, i.e., as a particular test of a particular feature variable in the feature vector $\boldsymbol{X} = (X_1, \cdots, X_J)$. All of these views are equivalent.

In some common cases, the decomposition $\Delta_{12} = \Delta_1 + \Delta_2$ is well known. Indeed, it has been glorified as a *Pythagorean theorem* [51]. For example, in the case of the squared error loss function, Δ_{12}, Δ_1, and Δ_2 are the "total sum of squares," the "between sum of squares," and the "within sum of squares," respectively. The best split on X is the one that maximizes the between sum of squares (i.e., the distance between left and right centroids), or equivalently, minimizes the within sum of squares (i.e., the residual variation).

In the case of the log likelihood loss function, Δ_{12}, Δ_1, and Δ_2 are the mutual informations $I(\boldsymbol{Y}; X \mid t)$, $I(\boldsymbol{Y}; \alpha(x) \mid t)$, and $I(\boldsymbol{Y}; X \mid \alpha(X), t)$, respectively. In this case, the best split on X is the one that maximizes the information $I(\boldsymbol{Y}; \alpha(X) \mid t)$ gained about $\boldsymbol{Y}$ when $\alpha(X)$ is learned, or equivalently, minimizes the information $I(\boldsymbol{Y}; X \mid \alpha(X), t)$ lost about $\boldsymbol{Y}$ when direct knowledge of X is replaced by knowledge of $\alpha(X)$ alone.

We will now show that the optimal α maps each outcome of X to its ***nearest neighbor***, $\mu(t_0)$ or $\mu(t_1)$, using the appropriate divergence as the distance measure. That is, the optimal binary split α has the following form:

$$\alpha(x) = \begin{cases} 0 & \text{if } d(x, \mu(t_0)) < d(x, \mu(t_1)) \\ 1 & \text{if } d(x, \mu(t_0)) > d(x, \mu(t_1)) \end{cases}$$

for all $x \in A$ if $P(x \mid t) > 0$, and is arbitrary if $P(x \mid t) = 0$ or if the divergences are equal. If this were not the case, i.e., if the optimal α were not a nearest neighbor mapping, then we could construct another mapping, say α', that would strictly decrease Δ_2 and hence strictly decrease the average impurity, contradicting the optimality of α.

To see this, assume $\alpha(x)$ is not a nearest neighbor mapping. Let $\beta(k) = \mu(t_k)$ be the centroid of $t_k = t \cap \{\alpha(X) = k\}$, as assumed in (12) and (13), and construct the nearest neighbor mapping

$$\alpha'(x) = \begin{cases} 0 & \text{if } d(x,\beta(0)) < d(x,\beta(1)) \\ 1 & \text{if } d(x,\beta(0)) > d(x,\beta(1)) \end{cases} \tag{14}$$

with $\alpha'(x)$ arbitrary if $P(x \mid t) = 0$ or if the divergences are equal. Since by assumption, $\alpha(x)$ is not a nearest neighbor mapping, there exists at least one x such that $P(x \mid t) > 0$, $d(x,\beta(0)) \neq d(x,\beta(1))$, and $\alpha(x) \neq \alpha'(x)$. Furthermore, for all such x's,

$$d(x,\beta(\alpha'(x))) < d(x,\beta(\alpha(x))).$$

Thus

$$\begin{aligned} \tilde{\Delta}_2 &= \sum_x P(x \mid t)\, d(x,\beta(\alpha'(x))) \\ &< \sum_x P(x \mid t)\, d(x,\beta(\alpha(x))) \\ &= \Delta_2, \end{aligned} \tag{15}$$

where the second equality follows from (13).

This does not yet prove that α' is a better partition than α, since β does not match α' and hence $\tilde{\Delta}_2$ is not a difference of average impurities, as is Δ_2. However, let $\beta'(k) = \mu(t'_k)$ be the centroid of $t'_k = t \cap \{\alpha'(X) = k\}$, so that

$$\begin{aligned} \beta'(k) &\triangleq \arg\min_{\hat{y}} E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t'_k] \\ &= \arg\min_{\hat{y}} \sum_{x:\alpha'(x)=k} P(x \mid t'_k) E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid x] \\ &= \arg\min_{\hat{y}} \sum_{x:\alpha'(x)=k} P(x \mid t'_k)\, d(x,\hat{\boldsymbol{y}}), \end{aligned} \tag{16}$$

where the last equality follows from the definition $d(x,\hat{\boldsymbol{y}}) = E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid x] - i(x)$. Then the difference between the average impurity of the intermediate partition induced by α' and the average impurity of the most refined partition is given, as in (12) and (13), by

$$\begin{aligned} \Delta'_2 &= \sum_x P(x \mid t)\, d(x,\beta'(\alpha'(x))) \\ &= \sum_k P(t'_k \mid t) \sum_{x:\alpha'(x)=k} P(x \mid t'_k)\, d(x,\beta'(k)) \\ &\leq \sum_k P(t'_k \mid t) \sum_{x:\alpha'(x)=k} P(x \mid t'_k)\, d(x,\beta(k)) \\ &= \sum_x P(x \mid t)\, d(x,\beta(\alpha'(x))) \\ &= \tilde{\Delta}_2. \end{aligned} \tag{17}$$

The inequality, of course, follows from (16).

Combining (15) and (17), we obtain

$$\Delta'_2 \leq \tilde{\Delta}_2 < \Delta_2. \tag{18}$$

In other words, the average impurity of α' is strictly lower than the average impurity of α, contradicting the fact that α is optimal. Hence the optimal α must be a nearest neighbor mapping.

We have proved the following for the binary $K = 2$ case.

Theorem (Optimal Partitioning): A necessary condition on any K-ary partition $A_0, \cdots, A_{K-1}$ of $A = \{x_1, \cdots, x_N\}$ minimizing the average impurity

$$I(A_0, \cdots, A_{K-1} \mid t) = \sum_{k=0}^{K-1} P(t_k \mid t) i(t_k)$$

is that $x \in A_k$ only if $k = \arg\min d(x, \mu(t_k))$, or if $P(x \mid t) = 0$, where $t_k = t \cap \{X \in A_k\}$.

Thus a necessary condition for a partition to minimize the average impurity is that its bins satisfy a nearest neighbor condition with their centroids, where the "distance" measure used for computing both the nearest neighbors and the centroids is the divergence corresponding to the given impurity measure. Here, of course, we have defined the impurity at node t to be the minimum expected loss

$$i(t) = \min_{\hat{y}} E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t],$$

the centroid of node t to be the output value achieving the impurity

$$\mu(t) = \arg\min_{\hat{y}} E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t],$$

and the divergence of an output value $\hat{\boldsymbol{y}}$ from the centroid $\mu(t)$ to be the excess expected loss

$$d(t,\hat{\boldsymbol{y}}) = E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t] - i(t).$$

The proof for $K > 2$ is a straightforward extension of the $K = 2$ case, and is not detailed here.

As an example, consider M-dimensional multivariate regression under the squared error loss function. As shown in Appendix A,

$$d(x,\mu(t_k)) = \|\mu(x) - \mu(t_k)\|^2,$$

where $\mu(x) = E[\boldsymbol{Y} \mid x]$ and $\mu(t_k) = E[\boldsymbol{Y} \mid t_k]$ are the expectations of the M-dimensional random vector $\boldsymbol{Y}$ given $t \cap \{X = x\}$ and $t \cap \{\alpha(X) = k\}$, respectively. According to the theorem, the optimal binary split α has the following form:

$$\alpha(x) = \begin{cases} 0 & \text{if } \|\mu(x) - \mu(t_0)\|^2 < \|\mu(x) - \mu(t_1)\|^2 \\ 1 & \text{if } \|\mu(x) - \mu(t_0)\|^2 > \|\mu(x) - \mu(t_1)\|^2 \end{cases}$$

with $\alpha(x)$ arbitrary if $P(x \mid t) = 0$ or if $\mu(x)$ is equidistant from $\mu(t_0)$ and $\mu(t_1)$. That is, assign x to the left child if the vector $E[\boldsymbol{Y} \mid x]$ is closer to $\mu(t_0)$ than to $\mu(t_1)$; otherwise assign x to the right child. This reduces to a simple hyperplane test: send x to the left whenever

$$\mu(x) \cdot (\mu(t_1) - \mu(t_0)) \leq (\|\mu(t_0)\|^2 + \|\mu(t_1)\|^2)/2,$$

where " $\cdot$ " denotes dot product. In one dimension, this reduces still further to a threshold test on $\mu(x) = E[Y \mid x]$, in accordance with Theorem 4.5 of Breiman *et al.* [7].

As another example, consider M-class classification under the log likelihood loss function. As shown in Appendix A,

$$d(x,\mu(t_k)) = D(\mu(x)\|\mu(t_k)),$$

which is the information divergence, or relative entropy, between M-dimensional probability mass functions $\mu(x)$ and $\mu(t_k)$ of $\boldsymbol{Y}$

given $t \cap \{X = x\}$ and $t \cap \{\alpha(X) = k\}$, respectively. Once again, the optimal α reduces to a simple hyperplane test on the probability vector $\mu(x) = E[Y \mid x] = P_{Y|X}(\cdot \mid x)$: send x to the left whenever

$$D(\mu(x)\|\mu(t_0)) \le D(\mu(x)\|\mu(t_1)),$$

or equivalently, whenever

$$\sum_{m=1}^{M} \mu_m(x) \log \frac{\mu_m(t_1)}{\mu_m(t_0)} \le 0.$$

In the two-class case, by expressing $\mu_0(x)$ as $1 - \mu_1(x)$, this reduces still further to a threshold test on $\mu_1(x) = P(Y = 1 \mid x) = E[Y \mid x]$, in accordance with [7, Theorem 4.5].

In fact, Burshtein *et al.* [48] have shown that the optimal binary α reduces to a simple hyperplane test on $\mu(x)$ for any loss function satisfying (6). This is the case for all of the loss functions listed in Appendix A.

Our theorem gives necessary conditions for a partition to minimize the average impurity, and hence greatly reduces the number of partitions that can possibly be optimal. In the two-bin case, if the nearest neighbor condition reduces to a hyperplane test, the number of partitions satisfying the necessary conditions for optimality is at most the number of linearly separable dichotomies of N points in M dimensions, or $C(N, M)$. (A dichotomy of N points into two sets A_0 and A_1 is *linearly separable* if there exists a hyperplane separating points in A_0 from points in A_1.) Cover [52] has shown that

$$C(N, M) = 2\sum_{m=0}^{M-1} \binom{N-1}{m} = \mathcal{O}(N^M).$$

Thus the theorem reduces the number of possibly optimal partitions from exponential in N to polynomial in N (but now exponential in M). To search through this number of possibly optimal partitions, Burshtein *et al.* [47] have proposed an $\mathcal{O}(N^M)$ time algorithm based on linear programming that is guaranteed to find the optimal partition in this case. Unfortunately, even in this case, a full search through the set of linearly separable dichotomies is infeasible, when either N or M is large. Thus there is still a need for an algorithm that can efficiently search through the space of potentially optimal partitions. We now develop such an algorithm. The algorithm will also work for $K \ge 2$ (the nonbinary case) and for arbitrary loss functions [i.e., loss functions that do not necessarily satisfy (6)].

Using our theorem, we can find the optimal partition $\alpha(x)$, if we know the centroids $\mu(t_k)$. Unfortunately, we can only find the centroids $\mu(t_k)$, if we know the partition $\alpha(x)$. The solution to this problem is to alternate these two steps, starting from an initial guess. It turns out that this procedure is a descent algorithm, in which the average divergence $\Delta_2 \ge 0$ is reduced at each step. Consequently, convergence is guaranteed. In fact, this algorithm is equivalent to the K-means algorithm [42] for determining pattern clusters or the generalized Lloyd algorithm [46] for designing vector quantizers, using the divergence as a distortion measure.

Formally, let $x_1, \cdots, x_N$ be N letters, each with weight $P(x \mid t)$, and let $A_0, \cdots, A_{K-1}$ be an initial partition of $A = \{x_1, \cdots, x_N\}$ into bins. For each bin A_k, let $\beta(k) = \mu(t_k)$ be its centroid, i.e., the vector $\hat{y}$ that minimizes $E[\ell(Y, \hat{y}) \mid t_k]$, or equivalently, minimizes $\sum_{x \in A_k} P(x \mid t_k)\, d(x, \hat{y})$, where $t_k = t \cap \{X \in A_k\}$. For many loss functions (e.g., those listed in Appendix A), $\beta(k)$ can be computed simply as the weighted arithmetic average $\beta(k) = \sum_{x \in A_k} P(x \mid t_k)\mu(x) = E[Y \mid X \in A_k]$. For others, $\beta(k)$ may have to be computed by a gradient search. Now, it follows from the construction of α' (14) and β' (16) in the proof of the theorem, and the result (18), that the following two step procedure reduces the average divergence Δ_2, and hence the average impurity (11).

1) Update α to α' for fixed β, by reassigning each x to its nearest neighbor in the divergence sense. That is, let $\alpha'(x) = \arg\min_k d(x, \beta(k))$, breaking ties arbitrarily.
2) Update β to β' for fixed α', by recomputing the centroid of each bin.

We have the following algorithm.

Algorithm (Partitioning): Iterate the above two steps to reduce the average impurity. When no further reduction occurs (after a finite number of iterations), quit. The final mapping α corresponds to a locally optimal partition.

Convergence is guaranteed because the algorithm is a descent algorithm. If a partition does not satisfy the necessary conditions for optimality, then a new partition is constructed with a strictly lower impurity. Except in the last iteration, a partition cannot be tried twice, because that would imply that the impurity does not strictly decrease between iterations. Hence the number of iterations is bounded by the number of possibly optimal partitions. As mentioned above, this is $\mathcal{O}(N^M)$ in the special but important case where the partition is binary and is given by a hyperplane. The general bound is $\mathcal{O}(K^N)$.

The computational complexity of each iteration of the partitioning algorithm is $\mathcal{O}(MKN)$, where, to repeat, M is the vector length of the centroids (e.g., the number of classes or the number of variables in multivariate regression), K is the number of bins (e.g., the number of children of a parent node), and N is the number of points to assign to the bins (e.g., the number of outcomes of the categorical variable X). The average number of iterations required is not known. Empirically, however, the number of iterations in K-means type algorithms is small (less than 20), and does not seem to depend heavily on M, K, or N [44, p. 99 ff.]. In contrast, the complexity of an exhaustive search is $\mathcal{O}(MK^N)$. Even the complexity of the Burshtein *et al.* algorithm, when $K = 2$ and the optimal partition is known to be a hyperplane, is $\mathcal{O}(N^M)$. Thus our K-means type algorithm can find a practically optimal partition in linear rather than exponential or polynomial time.

IV. Application to a Large Feature Alphabet

We have seen that the greedy growing algorithm for designing classification or regression trees seeks at each node the categorical feature variable and the partition of its alphabet that most reduces the overall risk of the tree. For a given node and feature variable, the partitioning algorithm of the previous section finds a locally optimal partition of the feature alphabet in time linear in the size of the alphabet. This permits the use of classification or regression trees in many problems where they would not otherwise be feasible, i.e., in those problems with large feature alphabets. In this section we examine one such problem: the letter-to-sound problem.

In the letter-to-sound problem we wish to translate sentences of text into strings of phonemes for use in a text-to-speech system. One way to solve the letter-to-sound problem is to treat it as a classification problem using a sliding window technique. In this technique, each character of text is classified into one of M phonemes by using as its feature vector a block of the immediately surrounding text. For example, to translate the text " ... `phonemes` ... " into the string of phonemes " ... `F-ONEM-Z` ... ", a window of length seven, say, slides over the text, and in each position, the classifier maps the text within the window to a phoneme that corresponds to the central character of the window. Characters not corresponding directly to a phoneme, such as silent "h"s, silent "e"s, spaces, and so forth, are ideally mapped into the null "-" phoneme, while characters that correspond to more than one phoneme, such as the "x" in "`exit`", are ideally mapped into special compound phonemes. The number of phonemic classes turns out to be $M = 59$ after the null phoneme and the special compound phonemes are added to the standard phoneme list. The number of letters in the feature alphabet is $N = 29$, since each feature variable is a character from the alphabet "a" through "z", plus *space, comma,* and *period.*

Sejnowski and Rosenberg solved this problem using an artificial neural network as the classifier [53]. Here, we solve the problem using a binary decision tree as the classifier. Actually, Lucassen and Mercer were the first to solve the problem using a binary decision tree. However, they were unable to handle the letters directly as feature variables, since the alphabet size N is too large to consider an exhaustive search (or for that matter the Burshtein *et al.* algorithm). Instead, they converted each window of seven characters to a 78-bit string of binary features, and used a conventional approach to tree design. In contrast, we handle the characters directly as feature variables by using the partitioning algorithm. In this way we are not constrained to use at each node one of, say, 78 preselected binary tests.

We implemented the partitioning algorithm for decision tree design by modifying the tree-growing software available with the S language statistical environment [54], [55]. As described in Appendix B, we smoothed the class probability density estimates at each leaf so that we could use the log likelihood loss function without the zeros in the histograms causing problems during clustering. The smoothing procedure is consistent with minimizing the log likelihood.

We collected training and test sets for the letter-to-sound problem from the NSF proposal used to initially fund this research [56]. The training set consists of a sequence of 5717 characters (39 sentences, 866 words) and the test set consists of a separate sequence of 1862 characters (13 sentences, 313 words). The training and test sequences were transcribed into phonemes by machine, and then the phoneme sequences were hand-aligned with the character sequences, by inserting null "-" phonemes and creating compound phonemes where appropriate.

Using the labeled training data, we designed a large classification tree and then pruned back to minimize the error rate according to a 10-fold cross validation [7], [10], [55]. The error rates of the resulting 168 leaf tree on both the training and test data are shown in Table II. Also shown in Table II are the error rates for Sejnowski and Rosenberg's neural network, on the same task. Unfortunately the comparison is crude because the training and test sequences are different in the two cases (although they have approximately the same length). However, preliminary results indicate that decision trees are an attractive alternative to neural networks in terms of classification accuracy. Moreover, in terms of computational complexity, decision trees have a tremendous advantage: only table lookups, no multiplications, are required to classify an object. The decision tree for this problem took only 20 minutes to design on a Sun Sparcstation 1, while the corresponding neural net took over 24 hours to design on a DEC VAX 11/780 with a floating point accelerator.

TABLE II
PROBABILITY OF ERROR (IN PERCENT) FOR THE LETTER-TO-SOUND PROBLEM

	Train	Test
Decision Tree	7%	10%
Neural Net	5%	22%

V. FURTHER APPLICATIONS

A. Variable Combinations

Suppose that we want to test more than one feature at a single node. This may be particularly desirable if the features are suspected of being individually uncorrelated with Y, but jointly highly correlated with Y. The problem of selecting a good n-tuple of features to test, and then finding a good binary test of those selected features, is usually dealt with heuristically [7], [57].

We can use the partitioning algorithm to help solve this problem, since we can treat each n-tuple as a single categorical variable with a large alphabet. For example, 10 binary feature variables can be combined into a single categorical feature variable with alphabet size 1024. The partitioning algorithm can then produce a map α that assigns each possible 10-bit pattern to either the left or right child. Conventional approaches, unable to deal with all 2^{1024} possible splits, restrict α to some simple form, such as a boolean product of binary tests, and its complement.

Although the partitioning algorithm does not directly address the selection of good n-tuples, it makes the selection process considerably easier by providing a fast algorithm for evaluating the merit of each candidate n-tuple.

B. Surrogate Splits

Suppose that at each node we wish to find not only the optimal split of the best feature variable X, but also the optimal split of the next best feature variable X'. This may be desirable if the tree is to handle missing data. If the primary feature variable X is somehow unavailable at a node, then we can use the second, *surrogate* split on the secondary feature variable X'.

To design a surrogate split when the optimal split on the primary variable is known, and the secondary variable is chosen, we can align the categorical outcomes of the secondary variable with the binary outcome of the optimal split. In [7], this is performed by a linear search for the split of the secondary variable that predicts with the minimum probability of error the binary outcome of the primary split. Unfortunately, that error is not connected in any way with the loss function of the problem. A better way is to assign the categorical outcomes of the secondary variable to either the left or right child, according to the nearest neighbor criterion.

However, it is possible to do even better, if we know the prior probability that the surrogate split is used, for then the surrogate split and the optimal split can be jointly designed. This idea is illustrated in Fig. 4. First, a decision is made as to whether to use the "optimal" or "surrogate" split, based, for example, on whether the primary feature variable is missing or not. The categorical outcomes of the primary and secondary feature variables are then aligned by assigning them to the left or right children so as to minimize the expected loss. This is done with the usual partitioning algorithm, treating the surrogate/optimal decision, the surrogate split, and the optimal split as one giant *composite node,* as in the figure. This composite node replaces the node

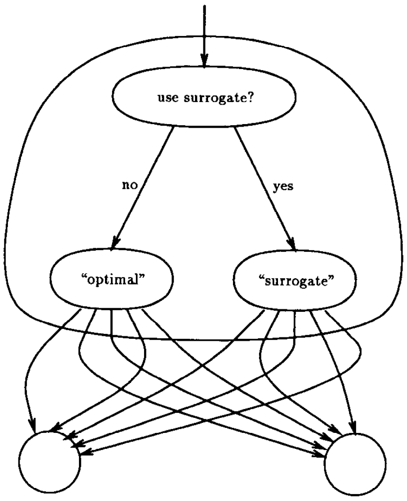

Fig. 4. Composite node for surrogate splits.

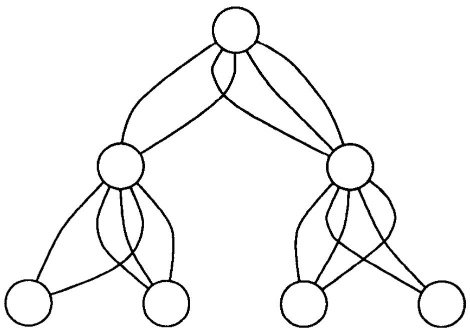

Fig. 5. The first two layers of a tree.

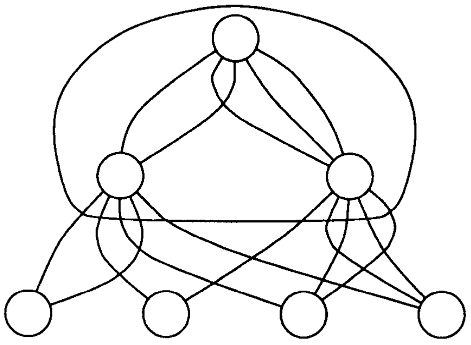

Fig. 6. The first two layers of a trellis.

in the original tree that contained only the optimal test. In this scheme, the "optimal" and "surrogate" tests are placed on almost equal ground.

C. Composite Nodes

As Fig. 4 suggests, it is possible to combine several tests into a single node, called a composite node. This arrangement may be an attractive alternative to variable combinations.

Composite nodes are easy to construct: simply build an ordinary tree; to terminate the tree in two children, treat the tree as a composite feature variable that has one outcome per leaf. The partitioning algorithm optimally assigns each outcome, or leaf, to one of the two child nodes.

There is no limit to this process. Composite nodes may consist of other composite nodes, which may consist of other composite nodes, etc. The *pylons* in the trees of Bahl *et al.* [9] are essentially constructed in this way, without the computational benefit of the partitioning algorithm.

D. Higher Order Splits and Decision Trellises

The optimal partitioning theorem and the partitioning algorithm apply just as easily to K-ary partitions, $K > 2$, as to binary partitions. The obvious implication of this is that they apply easily to the construction of K-ary trees, $K > 2$. A more interesting implication, however, is that they apply to the construction of directed acyclic decision graphs, which we call *decision trellises.*

We have already seen an example of a decision trellis: any tree containing a composite node, strictly speaking, is not a tree (because the children of the composite node have more than one parent within the composite node) but is a more general directed acyclic graph.

Things become more interesting if the leaves of the composite nodes are partitioned into more than two bins. For example, compare the ordinary tree shown in Fig. 5, constructed as usual by recursively applying the partitioning algorithm to the feature variable selected at each node, and the trellis structure shown in Fig. 6, constructed by treating the first layer as a composite node and applying the partitioning algorithm to its leaves, optimally assigning them to one of four bins. Clearly, the partition no longer respects the tree structure. However, the expected loss of the structure $E[\ell(Y, q(X))]$ is reduced, because the set over which the minimization occurs is less constrained. An experiment with the text-to-phoneme problem of Section IV shows that the reduction in expected loss (here, the class entropy) in the second layer of a seven-node trellis, such as the one in Fig. 6, is twice the reduction in expected loss in the second layer of a seven-node tree, such as the one in Fig. 5, when they have the same three internal nodes. See Table III.

TABLE III
REDUCTION IN CLASS ENTROPY (IN BITS) FOR THE LETTER-TO-SOUND PROBLEM

	Tree		Trellis	
Layer	Train	Test	Train	Test
1	0.16	0.16	0.16	0.16
2	0.08	0.10	0.20	0.19

By considering the first two layers as a composite node, the process can be repeated: the leaves of the composite node can be optimally assigned to eight bins, the leaves of that composite node can be optimally assigned to 16 bins, and so forth. In fact, there is no particular reason why the number of nodes at each layer must grow in powers of two. On the contrary, preliminary results suggest that it is advantageous to expand the trellis as quickly as possible at the top (e.g., split the root node N ways), and then expand slowly near the bottom. Unfortunately, since layer by layer, the procedure is still greedy, there is no guarantee that such a trellis structure will outperform a tree. Moreover, optimal pruning [7], [58] becomes impossible.

Nevertheless, trellises are richer in expressive power than trees. In a trellis, each node t has an interpretation as the disjunction (union) of a conjunction (intersection) of events, e.g., $t = (t_1 \cap \{X_1 \in A_1\}) \cup (t_2 \cap \{X_2 \in A_2\})$. In turn, each node t_1 and t_2 has a similar interpretation. In contrast, each node t of a decision tree has a conceptual interpretation of only a conjunction of events, e.g., $t = t_1 \cap \{X_1 \in A_1\}$, where t_1 has a similar interpretation. Thus the nodes of a decision trellis, generally speaking, organize themselves into higher conceptual representations than do the nodes of a decision tree. Consequently decision trellises may be more useful than decision trees in discovering the structure underlying a classification or knowledge representation problem.

VI. Summary and Conclusion

We presented a solution to the problem of finding the best K-ary partition of the outcomes of a discrete random feature variable, when the number of outcomes N is too large to consider an exhaustive search through the power set of possible partitions. In the Introduction we showed how finding such an optimal partition is required in the design of classification and regression trees, at each node and for each feature variable. In Section II we developed the framework for our solution, by generalizing Kullback's information divergence to divergences of arbitrary loss functions, and by showing their close connection to the impurity measures of Breiman *et al.* In Section III we presented our main results: a theorem on the necessary form of an optimal partition, and an iterative algorithm based on the theorem for finding a locally optimal partition in time per iteration linear in the size of the feature alphabet. In Section IV we applied the algorithm to a problem with a large feature alphabet, specifically, the problem of letter-to-sound conversion. In Section V we suggested further applications of the algorithm, including the design of variable combinations, surrogate splits, composite nodes, and directed acyclic decision graphs. The Appendix details impurity and divergence measures corresponding to a number of common loss functions, and shows how to smooth empirical probability distributions in the event that the training data are sparse.

The optimal partitioning theorem of Section III states that a necessary condition for a partition to minimize the expected loss is that its bins satisfy a nearest neighbor condition with their centroids, where the "distance" measure used for computing both the nearest neighbors and the centroids is the divergence corresponding to the given loss function. This theorem generalizes the corresponding Theorem 4.5 of Breiman *et al.* in several ways. First, whereas Breiman *et al.* show that a threshold condition is satisfied by *some* optimal partition, we prove that the threshold condition is actually necessary, and hence is satisfied by *every* optimal partition. Second, whereas Breiman *et al.* restrict themselves to binary partitions, we handle partitions with an arbitrary number of bins K. This is critical in the design of more complex decision graphs. Finally, and most importantly, whereas Breiman *et al.* restrict themselves to either binary classification or univariable regression, we handle arbitrary numbers of classes or arbitrary numbers of regression variables, M. The threshold of Breiman *et al.* thus generalizes to an $M - 1$ dimensional surface in an M dimensional space. For a number of common loss functions, including the squared error and the log likelihood, this surface is a simple hyperplane.

The algorithm is a K-means like clustering algorithm, which follows from the constructive proof of the theorem, and which, naturally, uses divergence in place of Euclidean distance. When the partitions are determined by hyperplanes, the computational complexity of the algorithm is only $\mathcal{O}(MKN)$ per iteration. Since the algorithm converges quickly, apparently with little dependence on M, N, or K, the overall computational complexity of the algorithm is linear in M, N, and K. This contrasts sharply with either the $\mathcal{O}(MK^N)$ complexity of an exhaustive search, or the $\mathcal{O}(N^M)$ complexity of the Burshtein *et al.* algorithm, which is applicable when $K = 2$ and when the loss function has a special form.

The reduction in computational complexity from exponential to linear in N and M permits the use of classification or regression trees in many problems where they would not otherwise be feasible. A good example of such a problem is letter-to-sound conversion, which was discussed in Section IV. In that problem, each feature is a character from a set of $N = 29$ possible letters. Whereas a computational complexity on the order of 2^{29} is infeasible, a computational complexity on the order of 29 is not only feasible; it is attractive. Moreover, the larger the number of letters N, the better behaved the K-means algorithm, since the expected loss as a function of the centroids becomes more "continuous," and the fixed points of the algorithm are unlikely to be degenerate. Thus our partitioning algorithm, though not guaranteed to find the optimal partition, complements the method of exhaustive search very nicely. For small N, exhaustive search can be used; for large N, the partitioning algorithm can be used.

How large N can be in practice depends primarily on the amount of training data available. Consider trying to use a decision tree to predict the next word in a sentence, based on the J previous words. The feature vector would consist of J categorical variables, with each variable having an alphabet size of N, the number of words in the vocabulary. The number of classes M also equals N. If $N = 10\,000$, say, then determining a locally optimal partition with our algorithm amounts to clustering ten thousand 10 000-dimensional histograms into two bins. The amount of data necessary to accurately estimate such histograms is well over one hundred million samples, which is nearly impossible to collect even with today's computer technology. For these problems, and even for much smaller problems such as the text-to-phoneme problem, "smoothing" of the probability density estimates is required, especially if the log likelihood loss function is used. (The log likelihood loss function does not permit any zeros in the probability densities.) We use a smoothing procedure described in Appendix B, which is consistent with minimizing the log likelihood loss.

Finally, it should be mentioned that since the partitioning algorithm is essentially just a clustering algorithm, the vast body of literature on clustering algorithms may be used to improve the algorithm's speed or performance. For example, a hierarchical clustering technique used in conjunction with k-d trees may be 20–50 times faster than straightforward techniques with little loss in performance [59].

Appendix A
Some Common Divergences

Weighted Squared Error (Regression)

Let y and $\hat{y}$ be real M-dimensional vectors, and let W be a real nonnegative definite $M \times M$ matrix. The weighted squared error loss function is defined

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = (\boldsymbol{y}-\hat{\boldsymbol{y}})'W(\boldsymbol{y}-\hat{\boldsymbol{y}}),$$

where $\boldsymbol{y}'$ denotes the transpose of $\boldsymbol{y}$. The centroid of an event t is the M-dimensional vector

$$\begin{aligned}\mu(t) &= \arg\min_{\hat{\boldsymbol{y}}} E\left[(\boldsymbol{Y}-\hat{\boldsymbol{y}})'W(\boldsymbol{Y}-\hat{\boldsymbol{y}}) \mid t\right]\\ &= E[\boldsymbol{Y} \mid t],\end{aligned}$$

and the impurity is the minimum value

$$\begin{aligned}i(t) &= E\left[(\boldsymbol{Y}-\mu(t))'W(\boldsymbol{Y}-\mu(t)) \mid t\right]\\ &= E[\boldsymbol{Y}'W\boldsymbol{Y} \mid t] - \mu'(t)W\mu(t).\end{aligned}$$

The divergence of $\hat{\boldsymbol{y}}$ from $\mu(t)$ is given by

$$\begin{aligned}d(t,\hat{\boldsymbol{y}}) &= E\left[(\boldsymbol{Y}-\hat{\boldsymbol{y}})'W(\boldsymbol{Y}-\hat{\boldsymbol{y}}) \mid t\right] - i(t)\\ &= (\mu(t)-\hat{\boldsymbol{y}})'W(\mu(t)-\hat{\boldsymbol{y}}),\end{aligned}$$

which is itself a weighted squared error. When W is the identity matrix and t is the whole space, we have the familiar relation

$$E\|\boldsymbol{Y}-\hat{\boldsymbol{y}}\|^2 = E\|\boldsymbol{Y}-\mu\|^2 + \|\mu-\hat{\boldsymbol{y}}\|^2.$$

Weighted Gini Index of Diversity (Classification)

Let $\boldsymbol{y}$ be an M-dimensional *class indicator vector,* let $\hat{\boldsymbol{y}}$ be an M-dimensional *class probability vector,* and let W be a real nonnegative definite $M \times M$ matrix. As with the weighted squared error, the loss function is defined

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = (\boldsymbol{y}-\hat{\boldsymbol{y}})'W(\boldsymbol{y}-\hat{\boldsymbol{y}}),$$

and the centroid of an event t, which minimizes the expected loss $E[\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) \mid t]$, is the M-dimensional probability vector

$$\mu(t) = E[\boldsymbol{Y} \mid t].$$

The impurity

$$\begin{aligned}i(t) &= E\left[(\boldsymbol{Y}-\mu(t))'W(\boldsymbol{Y}-\mu(t)) \mid t\right]\\ &= E[\boldsymbol{Y}'W\boldsymbol{Y} \mid t] - \mu'(t)W\mu(t)\end{aligned}$$

is known as the *weighted Gini index of diversity,* which reduces to

$$i(t) = 1 - \sum_m \mu_m^2(t)$$

in the unweighted case and still further to

$$i(t) = 2\mu_1(t)\mu_2(t)$$

in the unweighted two-class case [7]. The divergence of $\boldsymbol{y}$ from $\mu(t)$ is given by

$$\begin{aligned}d(t,\hat{\boldsymbol{y}}) &= E\left[(\boldsymbol{Y}-\hat{\boldsymbol{y}})'W(\boldsymbol{Y}-\hat{\boldsymbol{y}}) \mid t\right] - i(t)\\ &= (\mu(t)-\hat{\boldsymbol{y}})'W(\mu(t)-\hat{\boldsymbol{y}}),\end{aligned}$$

which is just a weighted squared error between probability mass functions.

Information Divergence (Classification)

Let $\boldsymbol{y}$ be an M-dimensional class indicator vector, and let $\hat{\boldsymbol{y}}$ be an M-dimensional class probability vector with nonzero components. The log likelihood loss function is defined

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = -\sum_m y_m \log \hat{y}_m,$$

which is thc approximate number of bits required to represent the class indicated by $\boldsymbol{y}$ using a Huffman code matched to the probability vector $\hat{\boldsymbol{y}}$. The expected loss given event t is the average number of bits required to represent the class indicated by $\boldsymbol{Y}$,

$$E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t] = -\sum_m \mu_m(t)\log \hat{y}_m,$$

which is minimized by the centroid $\mu(t) = E[\boldsymbol{Y} \mid t]$, the true probability mass function for Y given t. The impurity at node t is the value of the expected loss at the centroid,

$$i(t) = -\sum_m \mu_m(t)\log \mu_m(t),$$

which is just the entropy $H(\boldsymbol{Y} \mid t)$. The divergence of $\boldsymbol{y}$ from $\mu(t)$ is the average excess loss,

$$\begin{aligned}d(t,\hat{\boldsymbol{y}}) &= -\sum_m \mu_m(t)\log \hat{y}_m - i(t)\\ &= \sum_m \mu_m(t)\log\frac{\mu_m(t)}{\hat{y}_m}\\ &= D(\mu(t) \parallel \hat{\boldsymbol{y}}),\end{aligned}$$

or the relative entropy between the true distribution $\mu(t)$ and the arbitrary distribution $\hat{\boldsymbol{y}}$.

Weighted Misclassification Error (Classification)

Let $\boldsymbol{y}$ be an M-dimensional class indicator vector, let $\hat{\boldsymbol{y}}$ be an M-dimensional class probability vector, and let $W = (w_{mn})$ be a real $M \times M$ matrix whose mnth entry is the cost of representing the mth class by the nth class. If the true class distribution were indeed $\hat{\boldsymbol{y}}$, then the nth component of the row vector $\hat{\boldsymbol{y}}'W$ would be the expected cost of representing $\boldsymbol{Y}$ by the nth class. The best class to represent $\boldsymbol{Y}$ would then be

$$\hat{n}^* = \arg\min_n \sum_m \hat{y}_m w_{mn}.$$

The weighted misclassification error can now be defined

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = \sum_m y_m w_{m\hat{n}^*},$$

which is the cost of representing the class indicated by $\boldsymbol{y}$ by the class that minimizes the expected cost assuming the class distribution is $\hat{\boldsymbol{y}}$. The expected loss at node t is thus given by

$$E[\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) \mid t] = \sum_m \mu_m(t) w_{m\hat{n}^*},$$

where $\mu(t) = E[\boldsymbol{Y} \mid t]$ is the true class distribution at node t, and it is minimized, by the definition of $\hat{n}^*$, by $\hat{\boldsymbol{y}} = \mu(t)$, showing that $\mu(t)$ is indeed a centroid of node t. Note that the centroid is very nonunique. Any probability vector $\hat{\boldsymbol{y}}$ is a centroid provided that the $\hat{n}^*$ minimizing $\sum_m \hat{y}_m w_{mn}$ agrees with the n^* minimizing $\sum_m \mu_m(t) w_{mn}$. The impurity $i(t) = E[\ell(\boldsymbol{Y},\mu(t) \mid t)]$ is just the minimum possible expected loss if $\boldsymbol{Y}$ must be represented by just one class, and the divergence $d(t,\hat{\boldsymbol{y}}) = E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t] - i(t)$ is just the excess expected loss when the class used to represent $\boldsymbol{Y}$ is chosen as if $\hat{\boldsymbol{y}}$ were the distribution of $\boldsymbol{Y}$.

In the unweighted case ($w_{mn} = 0$ if $m = n$ and $w_{mn} = 1$ if $m \neq n$), $\hat{n}^*$ is the most probable class according to the probability vector $\hat{\boldsymbol{y}}$. Thus $\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) = 0$ if $\boldsymbol{Y}$ indicates class $\hat{n}^*$ and $\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) = 1$ otherwise. The expected value of this loss is equal to the probability of error when $\hat{n}^*$ is used to predict the class. This probability of error is minimized when $\hat{n}^* = n^*$, where n^* is the most probable class according to $\mu(t) = P_Y(\cdot \mid t)$. The impurity is thus the probability of error when $\hat{n}^* = n^*$,

$$i(t) = 1 - P_Y(n^* \mid t),$$

and the divergence is the increase in probability of error when $\hat{n}^* \neq n^*$,

$$d(t,\hat{\boldsymbol{y}}) = P_Y(n^* \mid t) - P_Y(\hat{n}^* \mid t).$$

Minimum Relative Entropy (Regression)

Let $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ be arbitrary real M-dimensional vectors. Given an M-dimensional vector function $\boldsymbol{f}$ on a random variable Z with reference measure R, we can define the loss function $\ell(\boldsymbol{y},\hat{\boldsymbol{y}})$ in terms of minimum relative entropies, as follows. Define

$$P_{\boldsymbol{y}} = \arg\min_Q \left\{ D(Q \parallel R) : \boldsymbol{y} = \int \boldsymbol{f}(z)\, dQ(z) \right\}$$

as the minimum relative entropy distribution between the reference measure R and the set of probability measures Q satisfying the expectation constraint $\boldsymbol{y} = \int \boldsymbol{f}(z)\, dQ(z)$. If no probability measures satisfy this constraint, then $P_{\boldsymbol{y}}$ is undefined. The relative entropy (also known as the Kullback–Leibler distance, discrimination information, and information divergence) between R and Q, when Q is absolutely continuous with respect to R, is defined

$$\begin{aligned} D(Q \parallel R) &= \int \log\left(\frac{dQ}{dR}\right) dQ(z) \\ &= \int q(z) \log \frac{q(z)}{r(z)}\, dz, \end{aligned}$$

where $q(z)$ and $r(z)$ are densities of Q and R with respect to some other reference measure, perhaps Lebesgue measure. (If R itself is Lebesgue measure, then $P_{\boldsymbol{y}}$ becomes the maximum entropy distribution satisfying the expectation constraint.) If Q is not absolutely continuous with respect to R, then $D(Q \parallel R)$ is defined to be infinite.

Likewise, let

$$P_{\hat{\boldsymbol{y}}} = \arg\min_Q \left\{ D(Q \parallel R) : \hat{\boldsymbol{y}} = \int \boldsymbol{f}(z)\, dQ(z) \right\}$$

be the minimum relative entropy distribution between R and the set of probability measures satisfying $\hat{\boldsymbol{y}} = \int \boldsymbol{f}(z)\, dQ(z)$. As can be seen by solving the variational equations, the minimum relative entropy distribution has a density of the form

$$p_{\hat{\boldsymbol{y}}}(z) = \frac{r(z)\exp\{\eta_{\hat{\boldsymbol{y}}}'\boldsymbol{f}(z)\}}{\int r(z)\exp\{\eta_{\hat{\boldsymbol{y}}}'\boldsymbol{f}(z)\}\, dz}, \tag{19}$$

where $\eta_{\hat{\boldsymbol{y}}}$ is an M-dimensional Lagrange multiplier chosen to satisfy $\hat{\boldsymbol{y}} = \int p_{\hat{\boldsymbol{y}}}(z)\boldsymbol{f}(z)\, dz$, and $\eta_{\hat{\boldsymbol{y}}}'$ is the transpose of $\eta_{\hat{\boldsymbol{y}}}$.

The loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ can now be defined (if it exists) by

$$\ell(\boldsymbol{y},\hat{\boldsymbol{y}}) = \int p_{\boldsymbol{y}}(z) \log(1/p_{\hat{\boldsymbol{y}}}(z))\, dz,$$

where $p_{\boldsymbol{y}}$ and $p_{\hat{\boldsymbol{y}}}$ are the densities of $P_{\boldsymbol{y}}$ and $P_{\hat{\boldsymbol{y}}}$.

If $\boldsymbol{y}$ is replaced by a random vector, say $\boldsymbol{Y}$, jointly distributed with Z, $P_{\boldsymbol{y}}$ becomes a conditional distribution of Z given $\{\boldsymbol{Y} = \boldsymbol{y}\}$. A marginal of Z, say $\overline{P}$, is induced by mixing the conditionals $P_{\boldsymbol{y}}$ by the distribution of $\boldsymbol{Y}$, which we assume is conditioned on node t. The expected loss can then be written

$$\begin{aligned} E[\ell(\boldsymbol{Y},\hat{\boldsymbol{y}}) \mid t] &= E\left[\int p_{\boldsymbol{Y}}(z)\log(1/p_{\hat{\boldsymbol{y}}}(z))\, dz \,\middle|\, t\right] \\ &= \int \overline{p}(z)\log(1/p_{\hat{\boldsymbol{y}}}(z))\, dz, \end{aligned} \tag{20}$$

where $\overline{p}$ is the density of $\overline{P}$. Remarkably, the expected loss is minimized by $\hat{\boldsymbol{y}} = \mu(t)$, where $\mu(t) = E[\boldsymbol{Y} \mid t]$, since by substituting (19) into (20) and taking partial derivatives with respect to η, we obtain

$$\begin{aligned} \frac{\partial}{\partial \eta} \int \overline{p}(z) &\left\{ \log \int r(w)e^{\eta'\boldsymbol{f}(w)}\, dw - \log r(z) - \eta'\boldsymbol{f}(z) \right\} dz \\ &= \frac{\int \boldsymbol{f}(z)r(z)e^{\eta'\boldsymbol{f}(z)}\, dz}{\int r(z)e^{\eta'\boldsymbol{f}(z)}\, dz} - \int \overline{p}(z)\boldsymbol{f}(z)\, dz \\ &= \hat{\boldsymbol{y}} - \mu(t), \end{aligned}$$

which equals zero when $\hat{\boldsymbol{y}} = \mu(t)$. Hence $\mu(t) = E[\boldsymbol{Y} \mid t]$ is the centroid of t, and $i(t) = \int \overline{p}(z)\log(1/p_{\mu}(z))\, dz$ is its impurity.

The above development is only formal, because the loss function is generally not integrable, and hence Fubini's theorem cannot be applied to the exchange of the expectation with the integral in the cross entropy (20). However, the divergence, or the difference between the expected loss and the impurity, is well defined,

$$\begin{aligned} d(t,\hat{\boldsymbol{y}}) &= \int \overline{p}(z)\log(1/p_{\hat{\boldsymbol{y}}}(z))\, dz - i(t) \\ &= \int \overline{p}(z)\log(p_{\mu}(z)/p_{\hat{\boldsymbol{y}}}(z))\, dz \\ &= (\eta_{\mu} - \eta_{\hat{\boldsymbol{y}}})'\mu(t) + \log\frac{\int r(z)e^{\eta_{\hat{\boldsymbol{y}}}'\boldsymbol{f}(z)}\, dz}{\int r(z)e^{\eta_{\mu}'\boldsymbol{f}(z)}\, dz} \\ &= \int p_{\mu}(z)\log(p_{\mu}(z)/p_{\hat{\boldsymbol{y}}}(z))\, dz, \end{aligned}$$

and the above arguments can be applied rigorously in this case [50], [60]. Note that the divergence is just the relative entropy between P_{μ} and $P_{\hat{\boldsymbol{y}}}$, which are in turn the minimum relative entropy distributions between the reference measure R and the constraint sets $\{Q : \mu(t) = \int \boldsymbol{f}(z)\, dQ(z)\}$ and $\{Q : \hat{\boldsymbol{y}} = \int \boldsymbol{f}(z)\, dQ(z)\}$.

Itakura–Saito Distortion (Regression)

Let $\boldsymbol{y} = (r_z(0), r_z(1), \cdots, r_z(M))$ and $\hat{\boldsymbol{y}} = (r_{\hat{z}}(0), r_{\hat{z}}(1), \cdots, r_{\hat{z}}(M))$ be Mth order autocorrelations of a discrete time stationary random process $Z = \{Z_n\}$, under two different process measures. Since $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are expectations of the same vector function $\boldsymbol{f}(\boldsymbol{z}) = (z_0^2, z_0z_1, \cdots, z_0z_M)$ under two different measures, the loss between them can be measured in terms of minimum relative entropy with respect to a reference stationary process measure R. Specifically, let R^n be the restriction of R to $\boldsymbol{Z}^n = (Z_0, Z_1, \cdots, Z_{n-1})$, $n > M$, and let

$$P_y^n = \arg\min_{Q^n} \left\{ D(Q^n \parallel R^n) : \boldsymbol{y} = \int \boldsymbol{f}(\boldsymbol{z}^n)\, dQ^n(\boldsymbol{z}^n) \right\}$$

be the minimum relative entropy distribution with respect to R^n satisfying $\boldsymbol{y} = \int \boldsymbol{f}(\boldsymbol{z}^n)\, dP_y^n(\boldsymbol{z}^n)$. Define $P_{\hat{y}}^n$ similarly. Then the loss, expected loss, centroid, impurity, and divergence are defined as usual, for example,

$$\mu(t) = E[\boldsymbol{Y} \mid t]$$

and

$$d^n(t, \hat{\boldsymbol{y}}) = D\left(P_{\mu(t)}^n \parallel P_{\hat{y}}^n\right). \tag{21}$$

It can be shown [61], [60] that if R is a zero-mean Gaussian autoregressive process, the *per letter* relative entropy $D\left(P_y^n \parallel P_{\hat{y}}^n\right)/n$ converges to one-half the Itakura–Saito distortion

$$D\left(P_y^n \parallel P_{\hat{y}}^n\right)/n \to d_{IS}(S_y, S_{\hat{y}})/2 = \int_0^\pi \left\{ \frac{S_y(\theta)}{S_{\hat{y}}(\theta)} - \log \frac{S_y(\theta)}{S_{\hat{y}}(\theta)} - 1 \right\} \frac{d\theta}{2\pi}$$

between the power spectral densities $S_y(\theta)$ and $S_{\hat{y}}(\theta)$ of the least squares Mth order linear predictive models with autocorrelation coefficients $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, respectively. It turns out that this is easy to compute [62], [34]:

$$d_{IS}(S_y, S_{\hat{y}}) = \frac{\boldsymbol{a}' W \boldsymbol{a}}{\sigma^2} - \log \frac{\hat{\sigma}^2}{\sigma^2} - 1,$$

where W is the Mth order autocorrelation matrix for a stationary process with Mth order autocorrelation vector $\boldsymbol{y}$, $\hat{\sigma}^2$ is the minimum Mth order linear prediction error for that process, $\boldsymbol{a}$ is the vector of optimal Mth order linear prediction coefficients for a process with Mth order autocorrelation vector $\hat{\boldsymbol{y}}$, and σ^2 is the gain for that process (assuming $a_0 = 1$). These quantities may be obtained by a standard LPC analysis, using Levinson's algorithm, for example [63].

We may now define a new divergence function based on the above limit:

$$d(t, \hat{\boldsymbol{y}}) = d_{IS}\left(S_{\mu(t)}, S_{\hat{y}}\right).$$

Clearly, this divergence is minimized by $\hat{\boldsymbol{y}} = \mu(t)$, as desired, since $\mu(t)$ minimizes (21) for every n.

Appendix B
Smoothing Empirical Distributions

The log likelihood loss function $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$ requires that all components in the probability vector $\hat{\boldsymbol{y}}$ be nonzero. Therefore, if the information divergence is used in the partitioning algorithm, the bin centroids must be "smoothed" to eliminate zeros. Nominally, each bin centroid $\mu(t_k)$ is the average of probability vectors $\mu(x)$, $x \in A_k$, and each $\mu(x)$ is in turn the empirical probability mass function (pmf) of $\boldsymbol{Y}$ given $X = x$. Thus $\mu(t_k)$ is nominally the empirical pmf of $\boldsymbol{Y}$ given $X \in A_k$. Empirical pmfs, or equivalently histograms, typically have many zeros, particularly if the number of classes M is large and the data are limited. There are a number of methods in the literature for "smoothing" empirical pmfs, which try to estimate the probabilities of unobserved events. These include Turing's formula [64] and Laplace's estimator [65]. We use another approach, similar to that of [9], which is consistent with our objective of finding the centroid that minimizes the average information divergence.

Precisely, suppose we are trying to estimate the centroids $\mu(t_0)$ and $\mu(t_1)$, where t_0 and t_1 are children of parent node t. Using the parent's centroid $\tilde{\mu}(t)$ as a prior, which we assume has already been likewise smoothed and hence contains no zeros, we choose as the smoothed centroid of each child the convex combination

$$\tilde{\mu}(t_k) = \lambda \mu(t_k) + (1 - \lambda)\tilde{\mu}(t),$$

where $\mu(t_k)$ is the unsmoothed centroid of t_k and $\lambda \in [0, 1)$ is chosen to minimize the total log likelihood loss over the entire data set,

$$\sum_j \ell(\boldsymbol{y}_j, \lambda \hat{\boldsymbol{y}}_j + (1 - \lambda)\tilde{\mu}(t)), \tag{22}$$

where for each sample $(x_j, \boldsymbol{y}_j)$ in the set with $x_j \in A_k$, $\hat{\boldsymbol{y}}_j$ is the unsmoothed centroid of the kth bin, computed as if the jth sample were not in the training set. (This is the "leave-one-out" estimate of the empirical pmf of $\boldsymbol{Y}$ given $X \in A_k$.) This jth sample is used instead in the "test" set. Summing the losses over the "test" set constructed in this way, we obtain (22), for a given λ.

We find the optimal λ in (22) by taking the derivative and using an iterative root finding algorithm. Conveniently, this can be done without going through the training set on every iteration, because a few histograms sufficiently summarize the data. Specifically, let $\tilde{P}(y)$ be the smoothed prior $\tilde{\mu}(t)$, let $P_j(y)$ be the "leave-one-out" estimate $\hat{\boldsymbol{y}}_j$, and let y_j be the class of the jth sample. The derivative of (22) with respect to λ becomes

$$-\sum_j \frac{d}{d\lambda} \log\left[\lambda P_j(y_j) + (1 - \lambda)\tilde{P}(y_j)\right] = -\sum_j \frac{P_j(y_j) - \tilde{P}(y_j)}{\lambda P_j(y_j) + (1 - \lambda)\tilde{P}(y_j)}. \tag{23}$$

We now split the sum according to whether $x_j \in A_0$ or A_1, and treat each component separately. Let $h(y)$ be the histogram of all L samples in bin k. Then $P_j(y_j) = [h(y_j) - 1]/(L - 1)$. Let $\Delta(y) = [h(y) - 1]/(L - 1) - \tilde{P}(y)$. Then the k component of (23) becomes

$$-\sum_{j: x_j \in A_k} \frac{\Delta(y_j)}{\lambda \Delta(y_j) + \tilde{P}(y_j)} = -\sum_y \frac{h(y)}{\lambda + \tilde{P}(y)/\Delta(y)},$$

which is easily computed at every iteration.

Acknowledgment

The author wishes to thank the following people for their contributions to this work: R. Gray of Stanford University, M. Riley of AT&T Bell Laboratories, D. Pregibon of AT&T Bell Laboratories, D. Nahamoo of IBM Watson Research Laboratories, J. Deken of the National Science Foundation, G. Groner of Speech Plus, Inc., E. Dorsey, formerly of Speech Plus, Inc., and P. Marks, formerly of Telesensory Systems, Inc.

References

[1] J. E. G. Henrichon and K. S. Fu, "A nonparametric partitioning procedure for pattern classification," *IEEE Trans. Comput.*, vol. C-18, pp. 604–624, May 1969.

[2] W. S. Meisel and D. A. Michalopoulos, "A partitioning algorithm with application in pattern classification and the optimization of decision trees," *IEEE Trans. Comput.*, vol. C-22, pp. 93–103, Jan. 1973.

[3] H. J. Payne and W. S. Meisel, "An algorithm for constructing optimal binary decision trees," *IEEE Trans. Comput.*, vol. C-26, pp. 905–916, Sept. 1977.

[4] P. H. Swain and H. Hauska, "The decision tree classifier: design and potential," *IEEE Trans. Geosci. Electron.*, vol. GE-15, pp. 142–147, July 1977.

[5] I. K. Sethi and B. Chatterjee, "Efficient decision tree design for discrete variable pattern recognition problems," *Pattern Recog.*, vol. 9, pp. 197–206, 1977.

[6] I. K. Sethi and G. P. R. Sarvarayudu, "Hierarchical classifier design using mutual information," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 441–445, July 1982.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees* (The Wadsworth Statistics/Probability Series). Belmont, CA: Wadsworth, 1984.

[8] J. M. Lucassen and R. L. Mercer, "An information theoretic approach to the automatic determination of phonemic baseforms," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, San Diego, CA, IEEE, Mar. 1984, pp. 42.5.1–42.5.4.

[9] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "A tree-based statistical language model for natural language speech recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1001–1008, July 1989.

[10] P. A. Chou, "Applications of information theory to pattern recognition and the design of decision trees and trellises," Ph.D. dissertation, Stanford Univ., Stanford, CA, June 1988.

[11] Y. Brandman, "Spectral lower-bound techniques for logic circuits," Comput. Syst. Lab., Stanford, CA, Tech. Rep. CSL-TR-87-325, Mar. 1987.

[12] R. W. Payne and D. A. Preece, "Identification keys and diagnostic tables: A review," *J. Roy. Stat. Soc. A.*, vol. 143, pp. 253–292, 1980.

[13] E. B. Hunt, J. Marin, and P. T. Stone, *Experiments in Induction.* New York: Academic, 1966.

[14] J. R. Quinlan, "Induction over large data bases," Heuristic Programming Project, Stanford Univ., Tech. Rep. HPP-79-14, 1979.

[15] ——, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[16] ——, "The effect of noise on concept learning," in *Machine Learning—An Artificial Intelligence Approach*, vol. II, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds. Los Altos, CA: Kaufmann, 1986, ch. 6, pp. 149–166.

[17] J. Cheng, U. M. Fayyad, K. B. Irani, and Z. Qian, "Improved decision trees: A generalized version of ID3," in *Proc. Fifth Int. Conf. Machine Learning*, Ann Arbor, MI, June 1988, pp. 100–107.

[18] J. R. Quinlan and R. L. Rivest, "Inferring decision trees using the minimum description length principle," *Inform. Computat.*, vol. 80, pp. 227–248, 1989.

[19] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, pp. 261–283, 1989.

[20] J. Mingers, "Empirical comparison of selection measures for decision tree induction," *Machine Learning*, vol. 3, pp. 319–342, 1989.

[21] M. Montalbano, "Tables, flow charts, and program logic," *IBM Syst. J.*, pp. 51–63, Sept. 1962.

[22] J. Egler, "A procedure for converting logic table conditions into an efficient sequence of test instructions," *Commun. ACM*, vol. 6, pp. 510–514, Sept. 1963.

[23] S. L. Pollack, "Conversion of limited-entry decision tables to computer programs," *Commun. ACM*, vol. 11, pp. 677–682, Nov. 1965.

[24] L. T. Reinwald and R. M. Soland, "Conversion of limited-entry decision tables to optimal computer programs II: Minimum storage requirement," *J. ACM*, vol. 14, pp. 742–755, Oct. 1967.

[25] D. E. Knuth, "Optimal binary search trees," *Acta Inform.*, vol. 1, pp. 14–25, 1971.

[26] K. Shwayder, "Conversion of limited-entry decision tables to computer programs—A proposed modification to Pollack's algorithm," *Commun. ACM*, vol. 14, pp. 69–73, Feb. 1971.

[27] A. Bayes, "A dynamic programming algorithm to optimise decision table code," *Australian Comput. J.*, vol. 5, pp. 77–79, May 1973.

[28] S. Ganapathy and V. Rajamaran, "Information theory applied to the conversion of decision tables to computer programs," *Commun. ACM*, vol. 16, pp. 532–539, Sept. 1973.

[29] H. Schumacher and K. C. Sevcik, "The synthetic approach to decision table conversion," *Commun. ACM*, vol. 19, pp. 343–351, June 1976.

[30] A. Martelli and U. Montanari, "Optimizing decision trees through heuristically guided search," *Commun. ACM*, vol. 21, pp. 1025–1039, Dec. 1978.

[31] 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*, vol. IT-28, pp. 565–577, July 1982.

[32] J. A. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *J. Amer. Statist. Assoc.*, vol. 58, pp. 415–434, 1963.

[33] A. Fielding, "Binary segmentation: The automatic interaction detector and related techniques for exploring data structure," in *Exploring Data Structures*, vol. I, C. A. O'Muircheartaigh and C. Payne, Eds. London: Wiley, 1977, ch. 8, pp. 221–257.

[34] A. Buzo, A. H. Gray Jr., R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 562–574, Oct. 1980.

[35] D. Y. Wong, B. H. Juang, and A. H. Gray Jr., "An 800 bit/s vector quantization LPC vocoder," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 770–780, Oct. 1982.

[36] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree structured source coding and modeling," *IEEE Trans. Inform. Theory*, vol. 35, pp. 299–315, Mar. 1989.

[37] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Inform. Processing Lett.*, vol. 5, pp. 15–17, May 1976.

[38] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: Freeman, 1979.

[39] W. D. Fisher, "On grouping for maximum homogeneity," *J. Amer. Statist. Assoc.*, vol. 53, pp. 789–798, Dec. 1958.

[40] G. H. Ball and D. J. Hall, "A clustering technique for summarizing multivariate data," *Behavioral Sci.*, vol. 12, pp. 153–155, Mar. 1967.

[41] E. W. Forgey, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications," *Biometrics*, vol. 21, no. 3, p. 768, 1965.

[42] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statistics and Probability*, vol. 1. Berkeley, CA: University of California Press, 1967, pp. 281–297.

[43] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[44] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[45] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 129–136, Mar. 1982; previously an unpublished Bell Laboratories Tech. Note, 1957.

[46] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, Jan. 1980.

[47] D. Burshtein, V. D. Pietra, D. Kanevsky, and A. Nádas, "A splitting theorem for tree construction," IBM, Yorktown Heights, NY, Tech. Rep. RC 14754 (#66136), July 1989.

[48] ——, "Minimum impurity partitions," *Ann. Stat.*, Aug. 1989, submitted for publication.

[49] P. Chou, "Using decision trees for noiseless compression," in *Proc. Int. Symp. Inform. Theory*, IEEE, San Diego, CA, Jan. 1990, abstract only.

[50] S. Kullback, *Information Theory and Statistics.* New York: Wiley, 1959; republished by Dover, 1968.

[51] B. Efron, "Regression and ANOVA with zero-one data: Measures of residual variation," *J. Amer. Statist. Assoc.*, vol. 73, pp. 113–121, Mar. 1978.

[52] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 326–334, 1965.

[53] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.*, vol. 1, pp. 144–168, 1987.

[54] R. A. Becker, J. M. Chambers, and A. R. Wilks, *The New S Language.* Pacific Grove, CA: Wadsworth & Brooks, 1988.
[55] M. H. Becker, L. A. Clark, and D. Pregibon, "Tree-based models," in *Statistical Software in S.* Pacific Grove, CA: Wadsworth, 1989.
[56] R. M. Gray, "Applications of information theory to pattern recognition and the design of decision tree classifiers," proposal to NSF Division of Information Science and Technology, IST-8509860, Dec. 1985.
[57] S. M. Weiss, R. S. Galen, and P. V. Tadepalli, "Optimizing the predictive value of diagnostic decision rules," in *Proc. Nat. Conf. Artificial Intelligence,* AAAI, Seattle, WA, 1987, pp. 521–526.
[58] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-constrained vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, pp. 31–42, Jan. 1989.
[59] W. Equitz, "Fast algorithms for vector quantization picture coding," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing,* IEEE, Dallas, TX, Apr. 1987, pp. 18.1.1–18.1.4.
[60] J. E. Shore and R. M. Gray, "Minimum cross-entropy pattern classification and cluster analysis," *IEEE Trans. Pattern Anal. Machine Intell.,* vol. PAMI-4, pp. 11–17, Jan. 1982.
[61] R. M. Gray, A. H. Gray Jr., G. Rebolledo, and J. E. Shore, "Rate-distortion speech coding with a minimum discrimination information distortion measure," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 708–721, Nov. 1981.
[62] R. M. Gray, A. Buzo, A. H. Gray, and Y. Matsuyama, "Distortion measures for speech processing," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-28, pp. 367–376, Aug. 1980.
[63] J. D. Markel and A. H. Gray, *Linear Prediction of Speech* (Communication and Cybernetics). New York: Springer-Verlag, 1976.
[64] A. Nadas, "On Turing's formula for word probabilities," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-33, pp. 1414–1416, Dec. 1985.
[65] J. Rissanen, "Complexity of strings in the class of Markov processes," *IEEE Trans. Inform. Theory,* vol. IT-32, pp. 526–532, July 1986.

Philip A. Chou (S'82–M'87) was born in Stamford, CT, on April 17, 1958. He received the B.S.E. degree from Princeton University, Princeton, NJ, in 1980 and the M.S. degree from the University of California, Berkeley, in 1983, both in electrical engineering and computer science, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1988.

Since 1977, he has worked for IBM, Bell Laboratories, Princeton Plasma Physics Lab, Telesensory Systems, Speech Plus, Hughes, and Xerox, where he was involved variously in office automation, motion estimation in television, optical character recognition, LPC speech compression and synthesis, text-to-speech synthesis by rule, compression of digitized terrain, and speech and document recognition. His research interests are pattern recognition, data compression, and speech and image processing. Currently, he is with the Xerox Palo Alto Research Center, Palo Alto, CA.

Dr. Chou is a member of Phi Beta Kappa, Tau Beta Pi, Sigma Xi, and the IEEE Computer, Information Theory, and Acoustics, Speech, and Signal Processing societies.