Decision-Tree Probability Modeling for HMM Speech Recognition

by

Jonathan Trumbull Foote

B.S.E.E., Cornell University, 1985
M.Eng., Cornell University, 1986

Thesis

Submitted in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy
in the Division of Engineering at Brown University

May 1994

This dissertation by Jonathan Trumbull Foote is accepted in its present form by
the Division of Engineering as satisfying the
dissertation requirement for the degree of
Doctor of Philosophy

Date......................... ....
Harvey F. Silverman

Recommended to the Graduate Council

Date......................... ....
Stephen E. Levinson

Date......................... ....
David B. Cooper

Approved by the Graduate Council

Date......................... ....

The Vita of Jonathan Trumbull Foote

Jonathan T. Foote was born in 1963 in Hollywood, California. He attended public schools in Santa Monica, California, somehow without learning how to surf. He received a Bachelor of Science (Electrical Engineering) degree in 1985 and a Master of Engineering (Electrical) degree in 1986 from Cornell University. From 1986 to 1988 he worked as a development engineer for Teradyne, Inc. in Boston, Massachusetts. From 1988 to 1993 he has been at Brown University working towards a Ph.D. in Electrical Engineering. He received an Outstanding Research Award from the Brown University Chapter of Sigma Xi in 1992, and a Brown Presidential Teaching Award in 1993. He is a member of IEEE, Sigma Xi, Computer Professionals for Social Responsibility, and the American Association for Engineering Education. He is also a professional musician, and enjoys not surfing in his free time.

IN the days when the white engineers were disputing the attributes of the feeder system that was to be, one of them came to to Enzian of Bleicheröde and said, "We cannot agree on the chamber pressure. Our calculations show that a working pressure of 40 atü would be most desirable. But all the data we know are grouped around a value of only some 10 atü."

"Then clearly," said the Nguarorerue, "you must listen to the data."

"But that would not be the most perfect or efficient value," protested the German.

"Proud man," said the Nguarorerue, "what are these data, if not direct revelation? Where have they come from, if not the Rocket which is to be? How do you presume to compare a number which you have only derived on paper with a number that is the Rocket's own?"

From *Tales of the Schwartzkommando*, ed. Steve Edelman

[Thomas Pynchon, *Gravity's Rainbow*]

Contents

List of Tables

List of Figures

You must bear in mind that the speech-game is, so to say, unpredictable.
I mean: it has no grounds. It is not reasonable (or unreasonable).
It is there—like our life.

Ludwig Wittgenstein

Chapter 1

Introduction

The process of automatic speech recognition (ASR) starts with a digitized representation of speech. Since this must be sampled several thousand times a second for reasonable fidelity, a typical short utterance needs several kilobytes of data to represent the same information (what was said) that could be written with a handful of ASCII characters. Clearly there is redundant, if not useless, information in the speech signal. The essence of speech recognition, or indeed any classification task, is to eliminate noise and redundancy, leaving only the true information content [87]. To do this, a good model of the underlying process is essential.

Hidden Markov models (HMMs) are a powerful and well-characterized method for speech recognition. HMMs offer a computationally reasonable way to model complex time-varying processes like speech; however, they rely on a set of assumptions about the speech process which may be inaccurate. It may be argued that over the past decade, progress in ASR has consisted primarily of tweaking and patching HMM systems to overcome some of their more egregious limitations. The work presented here continues in this proud tradition.

While there are a number of questionable HMM assumptions, the ones most pertinent to this work are as follows:

- A HMM is built and trained to maximize the likelihood that it generates a desired acoustic signal. Given different models, the one which generated the observed acoustic signal with the highest likelihood is considered to be the recognized word. Unfortunately, there is a fundamental mismatch between training and recognition: it is not important how well the model actually represents the speech signal! The only requirement is that the correct model be much more likely than the others, given the observed data. A good recognition strategy will not waste effort modeling every nuance of the speech signal; better performance may be obtained at less expense by modeling only those aspects of speech which are semantically useful, i.e. those that carry information about what was said. For example, the pitch of spoken English carries little information (except, perhaps, for second-order effects like prosodic cues) and is usually not modeled.

- In a typical HMM, the probability that a given acoustic vector is emitted by a certain state depends only on that state and is conditionally independent of the past. This is called the *independent-output assumption* and almost certainly does not reflect the true character of the speech signal, which at any instant is more-or-less correlated with preceding and following events.

- A HMM implicitly assumes that the choice of acoustic parameters is good. These are representations of the acoustic signal, chosen to preserve the semantic information in the speech signal while drastically reducing the dimensionality. (A typical parameterization encodes 20 ms of speech data into 26 parameters). The parameterization must be chosen *a priori* before any training or classification, and there no guarantee that the most popular parameterization is in any way optimal.

The work presented here describes an alternate method of probability estimation using decision trees. Using decision trees can mitigate the effects of the bad assumptions just described. Because trees are built in a supervised manner, useful variations are (hopefully) modeled while information-poor variability is not. The independent-output assumption is circumvented to some extent by considering the time-varying properties of the acoustic signal. Though this makes for a more complicated (higher-dimensional) feature space, the supervised training extracts the relevant information, and is shown to directly improve recognition performance. As a side-effect of the tree-based probability model, the relative importance of the individual acoustic parameters may be measured. This can lead to better parameterizations by providing a means to select those with higher importance.

The basic principles behind decision trees are discussed in Chapter 2, as well as the specific features used to construct tree-based probability models. Chapter 3 presents background information on HMMs, while Chapter 4 details how the tree-based models are trained and how they are integrated into the HMM system. Chapter 4 also summarizes similar previous work, and presents a brief analysis of computation and storage requirements of tree-based models relative to other types.

Chapter 5 presents experimental results on a connected-word talker-independent alphadigit recognition system at Brown University. The alphadigit set is generally regarded as the most difficult small-vocabulary set due to its many highly-confuseable words (e.g. the *E*-set *B*, *C*, *D*, *E*, *G*, *P*, *T*, *V*, *Z*) and word pairs (e.g. *A*–*T* vs. 8–*E*). Research at the Laboratory for Engineering Man-machine Systems (LEMS) is focused on the signal processing and acoustic modeling aspects of speech recognition, rather than language modeling. Consequently the alphadidgit task was chosen for both for its difficulty and its small size. A small-vocabulary system means that the intricacies of a language model can be dispensed with, yet good performance on the alphadigit task will require state-of-the-art feature extraction and acoustic modeling.

Once the whole world is divided the parts need names.
There are already enough names.
One must know when to stop.
Knowing when to stop averts trouble.

Tao Te Ching

Chapter 2

Decision Trees

This chapter discusses the principles and construction of decision trees. A decision tree is loosely defined as a hierarchical classification procedure determined by a sequence of questions. Once the first question has been asked, the choice of subsequent questions depends on the answer to the current question. This can be represented by a directed graph known as a tree. Figure 2.1 shows a decision tree. Questions are asked at "nodes;" the first question is asked at the "root node." The answer to a question posed at a particular node will select the appropriate "branch" to further nodes. A terminal node with no branches is known as a "leaf." The game "twenty questions" and a binary search are common examples of decision trees; they show how a small number of questions can select the right outcome (leaf) from among a very large number of possibilities.

Decision trees are a well-known and much-used weapon in the classification arsenal; they have been used for tasks as diverse as character recognition [24, 81, 22], associative searching [11], natural language modeling [4], and even medical diagnosis [18]. The types and methodologies of decision trees are diverse as the possible applications, therefore the discussion here will be restricted to the type most practical for the probability-estimation

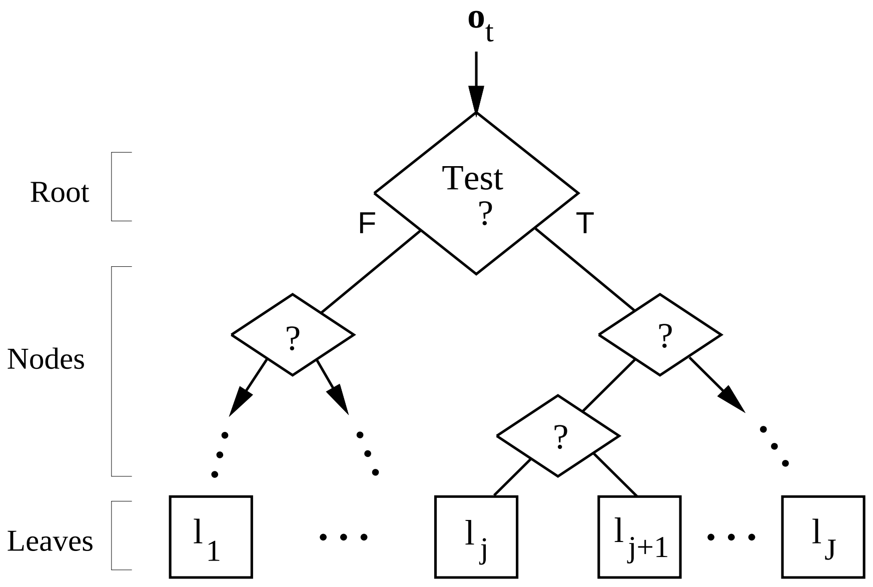

Figure 2.1: A decision tree.

task detailed in Chapters 3 and 4. This involves partitioning a high-dimensional feature space, which may be done using a particular tree structure called a K-d (K-dimensional) tree.

A particular vector must be on one side or another of a given hyperplane; this test can serve as the binary question at a node of a decision tree[1]. A more restricted test is to compare one dimension (element) of the vector with a scalar threshold; this produces a hyperplanar decision boundary perpendicular to the axis of the decision dimension. The feature space is divided into two regions by the test hyperplane; these regions may then be similarly subdivided in a recursive fashion. The end result is the feature space is partitioned into a number of non-overlapping *cells* (also called "buckets"), each of which correspond to a leaf of the tree. (Figure 3.4 illustrates such a partition for a two-dimensional space with one-dimensional tests.)

[1]Tests with more than two answers can be used, yielding a n-ary rather than a binary tree. In general, a single n-ary test can usually be accomplished with several binary tests. The hyperplane splits considered here have a naturally binary outcome, so discussion will be restricted to the binary case.

```
0. Start with all labeled data.
1. While stopping condition is unmet, do:
2. Find best split threshold over all thresholds and dimensions.
3. Send data to left or right child depending on threshold test.
4. Recursively repeat steps 1–4 for left and right children.
```

Table 2.1: Greedy tree construction algorithm.

2.1 Tree Construction

Because the construction of optimal decision trees is NP-hard [42], they are typically grown using a greedy strategy [18, 24, 6]. In practice, greedy trees grown in a top-down manner work quite well. The greedy algorithm (summarized in Table 2.1) works as follows: first, find the best split threshold t_d for all the data. This is the hyperplane normal to dimension d with d axis intercept t that best separates the data, according to some goodness measure called the *split criterion*. Thus t_d is found by maximizing the split criterion over all possible thresholds t in all possible dimensions d. (Computationally, this may be done with a brute-force search which is guaranteed to find the global maximum of the split criterion. A gradient-ascent approach will run much faster because only a small subset of possible thersholds are considered, hovever the global maximum may not be found.) The first threshold found corresponds to the first node in the classification tree. The left child then inherits the set of training samples less than the threshold $X_l : x_d < t_d$ while the right child inherits the complement. The splitting process is repeated recursively on each child, which results in further thresholds and nodes in the tree. At some point, a stopping rule decides that further splits are not worthwhile, and the splitting process is stopped. Three rules characterize a tree-growing strategy:

1. A *splitting rule* that determines where the decision threshold is placed, given the data in a node. For K-d trees, any linear combination of the input data may be used; often (as is the case here) only univariate splits are considered, resulting in decision boundaries normal to the coordinate axes.

2. A *stopping rule* that determines when the recursion ends. This is the rule that determines whether a node is a leaf node. (An alternate strategy is to use a lenient stopping rule to generate a very large tree, which may then be pruned to find a smaller and potentially better subtree.)

3. A *labeling rule* that assigns some value or class label to every leaf node. This is usually chosen to minimize some distance or distortion measure between the data in the leaf and a class label (classification) or functional estimate (regression). For the trees considered here, leaves will be associated (labeled) with the state-conditional output probabilities used in the HMM.

2.2 Splitting Rules

The splitting rule for a given node partitions the cell volume into two partitions, corresponding to the left and right children. In general, this can be any arbitrary partition. Because the space of possible partitions is enormous, finding the optimal general partition is not really practical (though work as been done using neural networks and other approaches, q.v. Section 4.6). Typically, the partition is constrained to be a hyperplane. Though probably suboptimal, the optimal hyperplane is straightforward both to find and use as a test. A further simplification is to only consider hyperplanes normal to a coordinate axis; thus only one feature dimension need be tested. Thus to find a split, each dimension is searched independently for a threshold (the hyperplane intercept) that maximizes the split criterion or "purity," which is some measure of how well the split separates the classes. Splitting a cell results in two subcells, each with its own class distribution; it is desired to split the cell so the two distributions are *as different as possible*. This naturally requires a distance metric, of which there exist a large variety (see [23] for a catalogue of *thirteen* different metrics). A useful and often-used metric is the mutual information, which will be used here exclusively.

2.2.1 Mutual Information

An excellent split criterion is the mutual information (MI) between the data and the class labels given the split. This metric has been proposed as early as 1962 [58], and has been used for tree-growing by several investigators [81, 1]. The MI is a quantitative measure of how much information a particular feature yields about the class it came from. For any split, the MI between the data and the classes may be easily calculated; the split that gives the maximum mutual information (MMI) is then selected as the decision for that node.

The mutual information is defined between different partitions of an underlying, discrete probability space, which may be thought of as the space of all possible outcomes of a given experiment. Let X denote a discrete partition of the feature space corresponding to a set of experimental outcomes (e.g. VQ codes or tree leaves), while C represents the partition corresponding to the class labels of the outcomes (e.g. the phonetic class of each VQ code or tree leaf). If the joint probability that a certain event $x_j \in X$ has class label $c_i \in C$ is $\Pr(c_i, x_j)$, then the mutual information (MI) between the events and the pattern classes is

[27]

$$\begin{aligned}
I(X;C) &\triangleq \sum_i \sum_j \Pr(c_i, x_j) \log_2 \frac{\Pr(c_i, x_j)}{\Pr(c_i)\Pr(x_j)} && (2.1)\\
&= H(X) - H(X|C) && (2.2)\\
&= H(C) - H(C|X) && (2.3)\\
&= \sum_i \sum_j \Pr(x_j, c_i) \log_2 \frac{\Pr(x_j|c_i)}{\Pr(c_j)} && (2.4)
\end{aligned}$$

where $H(\cdot)$ is the Shannon entropy function, in bits:

$$H(X) = -\sum_i \Pr(X = x_i) \log_2 \left(\Pr(X = x_i)\right) \tag{2.5}$$

and the conditional entropy $H(X|Y)$ is

$$\begin{aligned}
H(X|Y) &= -\sum_j \Pr(Y = y_j) \sum_i H(X|Y = y_j) && (2.6)\\
&= -\sum_j \Pr(Y = y_j) \sum_i \Pr(X = x_i|Y = y_j) \log_2 \left(\Pr(X = x_i|Y = y_j)\right). && (2.7)
\end{aligned}$$

Note that if the distributions are independent, then $\Pr(x_j, c_i) = \Pr(c_i)\Pr(x_j)$ so the log term in Equation 2.1 becomes zero. Thus the mutual information between independent random variables is zero. From the classification viewpoint, this is the common-sense statement that a data distribution independent of the classes gives no information about the classes.

To calculate the mutual information of a split, consider a threshold t_d in dimension d. This will split the data X into two partitions $X = \{Xa, Xb\}$, such that

$$Xa: \quad x_d \geq t_d \tag{2.8}$$

$$Xb: \quad x_d < t_d \tag{2.9}$$

i.e. Xa is the set of data points such that the dth dimension of the feature x is above the threshold and Xb the set of points where it is below. The mutual information from a partition $I(X;C)$ given any t and d is easily estimated from training samples in the following manner. Over the volume of the current node, count the relative frequencies:

$$N_j = \text{Total number of data points in cell } j \tag{2.10}$$

$$NC_{ij} = \text{Number of data points in cell } j \text{ from class } i \tag{2.11}$$

$$NXa_i = \text{Number of data points from class } i : x_d \geq t_d \tag{2.12}$$

$$NXb_i = \text{Number of data points from class } i : x_d < t_d \tag{2.13}$$

In the region, define $Pr(c_i)$ to be the probability of class i and $Pr(Xb_i)$ as the probability that a member of class i is below the given threshold. These probabilities are easily estimated as follows: (for clarity of notation, conditioning on the threshold is not indicated)

$$\Pr(c_i) \approx \frac{NC_{ij}}{N_j} \tag{2.14}$$

$$\Pr(Xb_i) \approx \frac{NXb_i}{NC_{ij}} \tag{2.15}$$

With these probabilities, the mutual information given the threshold is [27]

$$\begin{aligned} I(X;C) &= H(C) - H(C|X) && (2.16)\\ &= -\sum_i \Pr(c_i)\log_2 \Pr(c_i) + \sum_i \Pr(c_i) H_2\left(\Pr(Xb_i)\right) && (2.17)\\ &\approx -\sum_i \frac{NC_{ij}}{N_j}\log_2 \frac{NC_{ij}}{N_j} + \sum_i \frac{NC_{ij}}{NX} H_2\left(\frac{NXb_i}{NC_{ij}}\right) && (2.18)\end{aligned}$$

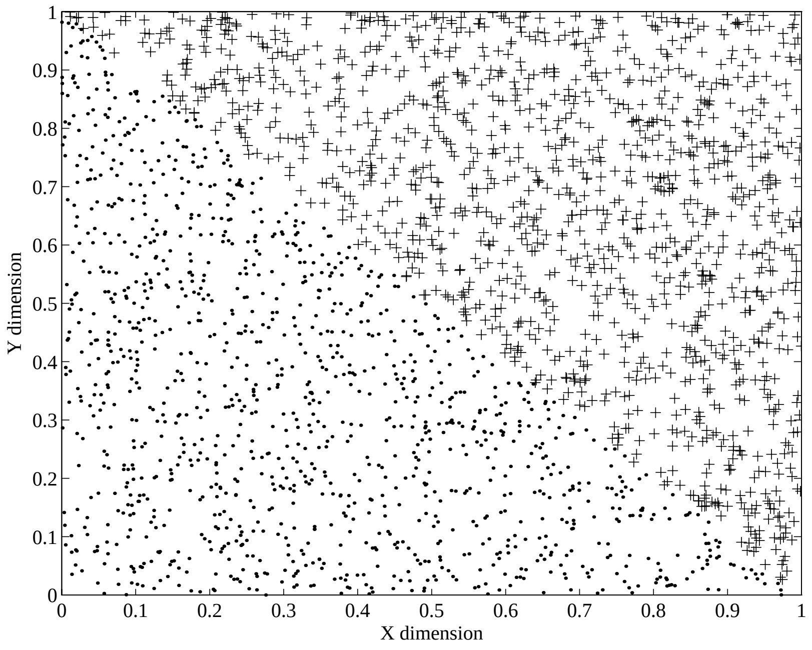

Figure 2.2: Example two-class distribution.

where H_2 is the binary entropy function

$$H_2(x) = -x \log_2(x) - (1-x)\log_2(1-x). \tag{2.19}$$

This is a convex function of the threshold value, and is also less than or equal to one (one bit is the most information that can be obtained by a binary split).

2.2.2 A Simple Example

For the purposes of demonstration, consider the two-class distribution of Figure 2.2. The class-independent distribution is uniform on the unit square. Data points are labeled as class zero ("·") if the sum of the x and y coordinates is less than one or class one ("+") if greater. The two classes are linearly separable, but the decision boundary is not parallel to a coordinate axis. A tree-based classifier can nonetheless approximate the decision boundary

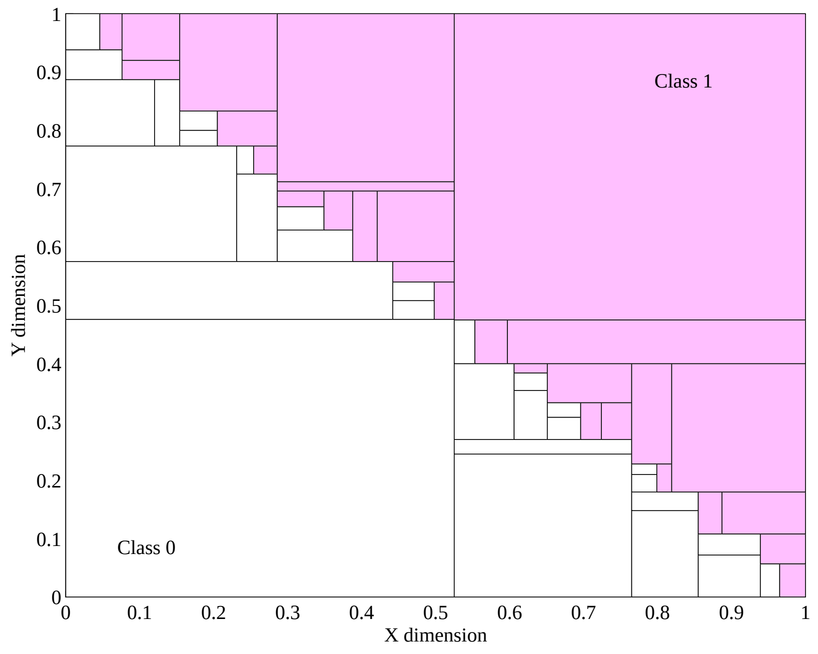

Figure 2.3: Classification tree partition.

to arbitrary accuracy, even using decision planes constrained to be normal to coordinate axes, as shown in Figure 2.3. (Of course, linear-discriminant analysis could find a projection making this distribution separable with one decision boundary[2]; the point is that a decision tree can do *equally badly* on *any* possible distribution, regardless of decision boundary shape.) Here, the leaf cells are colored according to the most probable class in the cell, i.e., the class with the most data points in that cell.

Figure 2.4 shows how the MMI splitting threshold is determined. In a two-class problem, Equation 2.3 may be written

$$I(X;C) = H(X) - \Pr(c_0)H(X|c_0) - \Pr(c_1)H(X|c_1). \tag{2.20}$$

For the distribution of Figure 2.2 projected onto dimension X, these four quantities are

[2]Real-world distributions are rarely so accommodating.

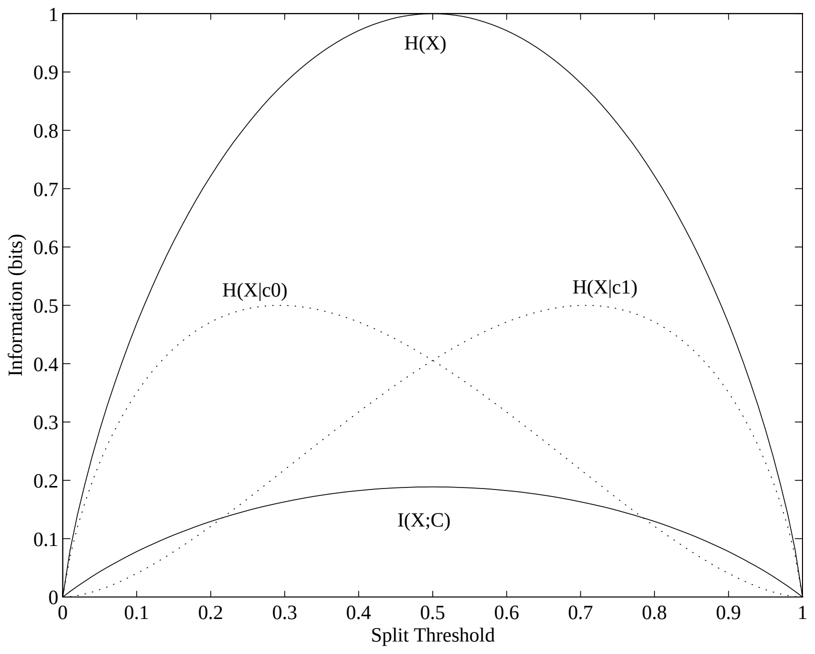

Figure 2.4: Entropy and information vs. split threshold.

plotted as a function of the threshold in Figure 2.4. Here, the action of the various terms may be seen. Because the total density is uniform over [0,1], $H(X)$ is the familiar binary entropy function (equation 2.19). This is maximum at the center of gravity of the class-independent distribution, regardless of its form; i.e. that split which divides the cells into two equal-probability volumes. (This is a nice feature as it tends to keep the tree relatively balanced: the two halves of a split node will tend to have roughly equal probability masses). Subtracted from $H(X)$ are the two class-conditional entropies $H(X|c_0)$ and $H(X|c_1)$, weighted by the relative priors. These are maximized at the center of mass of the class-conditional distributions, but since these terms are subtracted, they tend to bias the threshold *away* from the means of the classes, while $H(X)$ forces the threshold towards the center of the total distribution. Note that the mutual information is maximized at a threshold of 0.5; this corresponds to the vertical line in the center of Figure 2.3, which is the first split in the decision tree.

2.2.3 Dependence of MMI Split on Class Priors

Consider the distribution of Figure 2.2 with unequal class priors; i.e. one class has more members than another. Projected onto a single dimension, this results in the pdf shown in Figure 2.5, where the prior probability of a data point coming from class 0 is p and from class 1 is $(1-p)$. The Bayes threshold is the threshold that minimizes the classification error, shown as the crosshatched area in Figure 2.5. Contrary to statements published elsewhere [81], the threshold that maximizes $I(X;C)$ is not necessarily the threshold that minimizes the Bayes probability of error. Figure 2.6 shows the threshold variation as a function of class prior probabilities for the two-class distribution of Figure 2.5. The Bayes threshold is very dependent on the class priors, while the MMI threshold is nearly insensitive to them. Some may be distressed by the fact that the MMI boundary does not minimize classification error; this is certainly distressing if the tree is used to discriminate between classes with known prior probabilities. For some applications, however, the priors are *not* known, in which case the MMI boundary will be more robust if class priors vary significantly. For speech recognition, class prior probabilities should be determined by the language model, not the acoustic model, which ideally should not be biased by the relative frequencies of classes in the training data [66]. In any case, the trees are not (primarily) used as classifiers but rather to partition the feature space into regions that are meaningfully acoustically different. For this task, the boundaries should be relatively insensitive to the prior probabilities, as the MMI criterion will ensure.

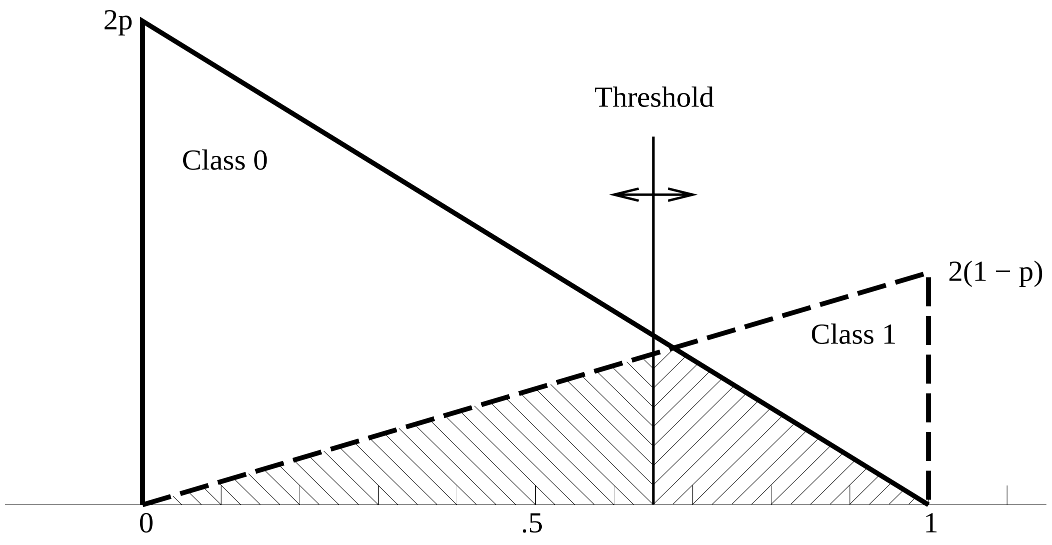

Figure 2.5: Example probability density functions.

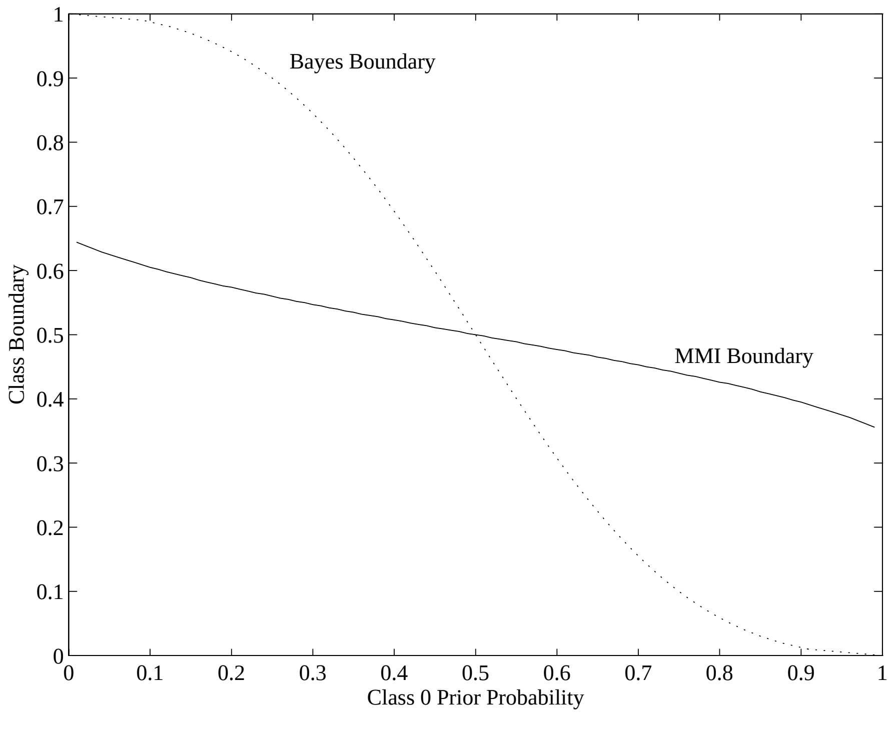

Figure 2.6: Threshold vs. class prior probabilities.

2.3 Stopping Rules

A second part of a tree growing strategy is the stopping criterion. This is a rule or set of rules that define the termination condition for the recursive splitting. An effective stopping rule is to terminate splitting when the number of training points in the cell falls below some threshold, but this rule may be too simplistic because it does not consider the goodness of the possible splits. On the other hand, the MMI criterion of Section 2.2.1 works well for finding good splits, but is a poor stopping condition because it is generally non-decreasing. (Imagine a tiny cell containing only two data points from different classes: splitting the cell so that each side gets a class yields an entire bit of mutual information. Bigger cells with overlapping distributions have less mutual information.) This motivates other ways of finding when further splitting is not worth the effort. In practice, trees are commonly grown quite large, and then pruned to the desired size, so the stopping criterion is not especially critical.

2.3.1 Delta-Entropy

One of the simplest stopping criteria is to look at the difference between the class entropy of the current node and that of its parent. The class entropy $H(C)$ is simply the entropy function of the (discrete) class distribution of the node.

$$H(C) = -\sum_i \Pr(c_i) \log_2 \Pr(c_i) \tag{2.21}$$

If the class entropy $H(C)$ from the best split of this node is less than some fraction of the parental best-split $H(C)$, then the node is considered a leaf and splitting is stopped. (Typically, stopping and pruning thresholds like the delta-entropy fraction are determined empirically, by adjusting the threshold until a tree of desired size is obtained.)

2.3.2 Mass-proportional mutual information

If the number of points in a node is small, the probability estimates for that node will tend to be unreliable. This motivates a stopping metric where the best-split mutual information is weighted by the probability mass inside the cell l_j to be split:

$$\text{stop}(l_j) = \left(\frac{N_j}{N}\right) I_j(X;C) \tag{2.22}$$

Further splits are not considered when this metric falls below some threshold. This criterion has the benefit that it excludes splits that might be suspect because of insufficient data points in the cell. The mass-weighted MMI criterion thus insures that splitting is not continued:

- If the split criterion is not large enough.
- If the probability mass in this bin is small enough that there are too few points to reliably estimate the split criterion.

The mass-weighted MMI criterion has some additional nice properties:

- It is a strictly decreasing function of depth in the tree, so that a fixed threshold will always yield a finite tree, regardless of the data distribution.
- It facilitates a simple tree pruning algorithm.
- It is very simply computed, by Equation 2.22.
- It preserves dimensional importance, as discussed in the next section (2.3.3).

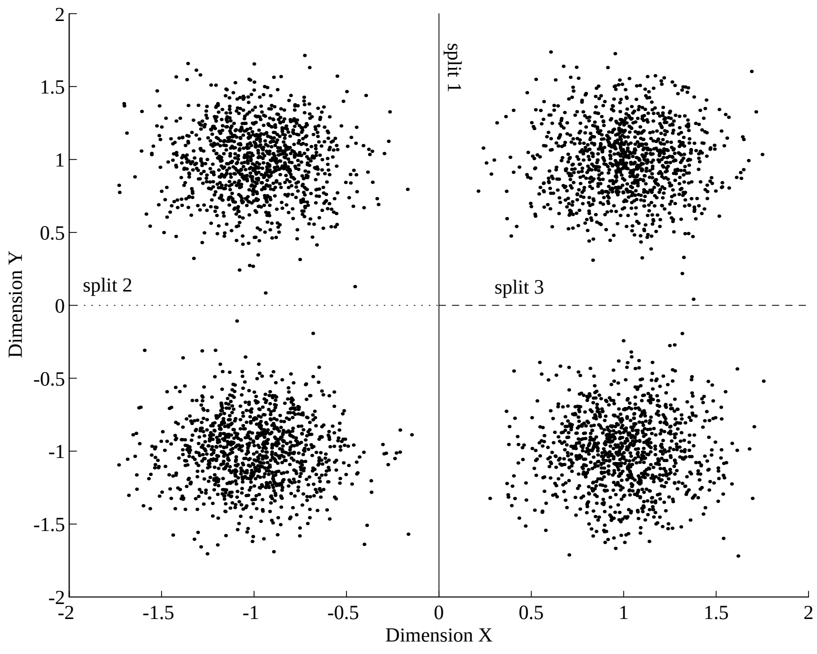

Figure 2.7: Four-class splitting example.

2.3.3 Dimensional Ranking

An interesting aspect of information trees is that they can give valuable information about the relative importance of each dimension in the feature space. If a given dimension is never used for a split, then it can be assumed that that dimension yields little or no information about the classes, and could be ignored with little loss. Conversely, if most splits occur in a few dimensions, then those dimensions are most important to the classifier, and could be emphasized.

It is not obvious how to accurately quantify the dimensional information yielded by the splits. A naive approach might be to simply divide the number of splits in a given dimension by the total number of splits, but this conflicts with the intuition that splits lower in the tree yield less information than ones higher up. Slightly better is to consider the mass-weighted MMI gain summed across all splits in a dimension. This works well for simple cases. Consider a two-dimensional problem with four equiprobable, separable classes as in Figure 2.7.

The entropy of the classes is two bits ($\log_2(4)$), and because they are separable, the mutual information between the data and the classes should be two bits as well. (Given a data point, we can find the class it came from with no uncertainty.) Arbitrarily, do the first split vertically. This gives one bit of mutual information (the best you can do with a binary split). This makes sense—given the result of the split test, we have reduced our uncertainty about the classes from four equiprobable classes to two. Now the next split also gives one bit of information, but because the cell only has half the data, the metric is one half bit. Similarly, the third split also yields one half bit of information, so the total metric is two bits, which agrees with intuition. Similarly, both dimensions give an equal amount (1 bit) of information, and are therefore equally important for classification.

2.4 Pruning

An often-used alternative to a given stopping criterion is to build a large tree and to "prune" it to a smaller size. This is done by changing internal tree nodes to leaves, thus ignoring further splits at a particular node. Pruning can result in a more optimal tree of a given size because a low-information node may have high-information children; in a top-down approach growth would stop at that node but in bottom-up pruning the contribution of the children would be recognized and the node kept. In the context of speech recognition, pruning allows the size of the trees, and hence the number of free parameters in the probability model, to be easily changed.

Optimal tree pruning is analogous to the canonical "0-1 knapsack" problem [26]. As such, the optimal solution requires a dynamic-programming approach to find the desired

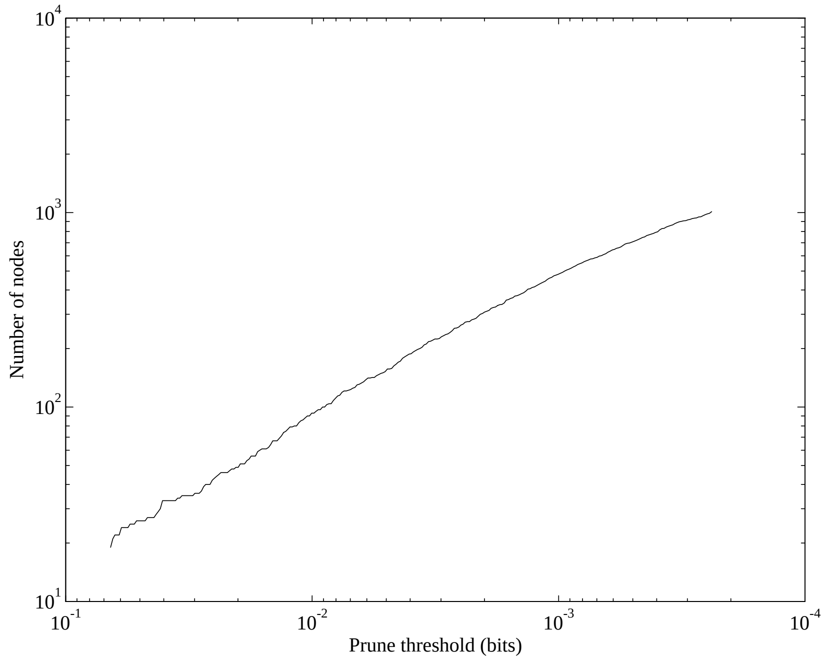

Figure 2.8: Tree size vs. pruning threshold.

number of leaves with the best aggregate cost function. In practice, there are a large number of leaves with similar costs, so a greedy approach will quickly yield a subtree very near, if not identical to, the optimal subtree. A simple greedy pruning algorithm starts with the full tree, where each node in the tree has been labeled with the sum of the cost of all children. A pruning threshold is initialized to the cost of the least-important leaf. As the pruning threshold is increased, nodes with aggregate costs below the threshold are pruned until a tree of the desired size remains. Fig. 2.8 demonstrates the tree size as a function of the pruning threshold. Using the mass-weighted MMI criterion gives a nice empirical relationship: the number of nodes is roughly equal to the reciprocal of the threshold. This makes it simple to prune a tree to the desired size.

Another widely-used pruning approach is to make a cost function that depends on both the size of the tree and the classification error. This is analogous to *rate distortion coding*, where the size of the tree corresponds to the code size (the rate) and the classification error corresponds to the distortion. A bigger tree means better classification, and there is some optimal tree size for the desired classification error (the achievable rate for the desired distortion), which may be found by minimizing the aggregate cost function. This approach is not suited to the problems considered here, because minimum classification error and tree size are not really relevant to the probability estimation task.

2.5 Labeling Rules

A decision tree may be used for classification by "labeling" each leaf with an approprate class label. If a value is used instead of a label, the tree may be used for regression or function estimation. Decision trees will be integrated with hidden Markov models by using them to estimate the probability that a given state emitted a certain observation. Thus with each leaf will be "labeled" with an array of "output" probabilities. The details of this procedure require an explanation of hidden Markov models, and so will be deferred until Chapter 4. In Chapter 5, the trees will also be used as classifiers, by labeling each leaf with the most probable class among the data points in that leaf. Once again, discussion will be postponed until Section 5.7.1.

...we shall regard all that is presented in this volume, and all that shall follow, as merely a set of conceptual and mathematical tricks, a bit of scientific legerdemain, which, for reasons unknown, but for which we are profoundly grateful, prove to be startlingly more successful in many applications than anyone who examines the basic assumptions would have any right to expect.

Richard Bellman, Adaptive Control Processes

Chapter 3

Hidden Markov Models

Figure 3.1 shows a stereotypical hidden Markov model (HMM) for a speechlike random process. An HMM is a system which has S distinct states $\{s_1, s_2, \ldots, s_S\}$. At regular time intervals ("ticks"), a state s_i must transition to another state s_j with probability a_{ij}

$$a_{ij} = \Pr(q_{t+1} = s_j | q_t = s_i) \tag{3.1}$$

where q_t is the state occupied at time t. Because the state transitions and outputs depend only on the current state, the model is (first-order) Markov. A path through the HMM may be denoted as $Q = \{q_1, q_2, \ldots\}$, again where q_t is the state s_i occupied at time t. At every time t, a state s_i emits an "observation" $\mathbf{o}_t$ with a state-conditional probability $\Pr(\mathbf{o}_t | q_t = s_i)$. (An alternate formulation has outputs emitted by the state transitions, or "arcs"—the two forms are equivalent.) Because actual state sequences (the actions above the wavy line) are not directly observable, the model is termed "hidden." This model was introduced by Baum [8] and popularized for speech recognition by work at IBM [49] and ATT [57].

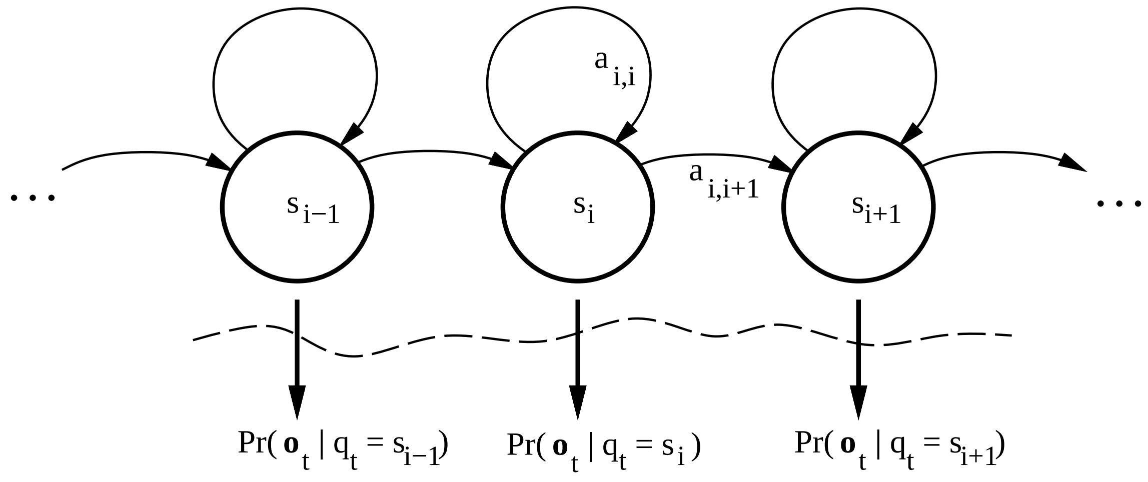

Figure 3.1: Hidden Markov model.

A HMM λ is characterized by the initial state probability distribution $\boldsymbol{\pi}$, the state-to-state transition matrix A, and the observation-output probability matrix B. The B matrix accounts for the probability that the observation $\mathbf{o}_t$ was emitted by state s_i at time t, denoted as

$$b_i(\mathbf{o}_t) = \Pr(\mathbf{o}_t|q_t = s_i) \tag{3.2}$$

HMMs are perhaps the most successful speech-recognition approach to date, and there exist well-developed algorithmic methods for solving the following fundamental problems (once a model topology has been determined *a priori* and training data obtained). These are:

1. Training: given a known sequence of observations (training data) $\mathbf{O}$, adjust the HMM model parameters $\lambda(\boldsymbol{\pi}, A, B)$ to maximize $\Pr(\mathbf{O}|\lambda)$.

2. Recognition: given a sequence of observations $\mathbf{O}$ and a trained model λ, find the most "optimal" state sequence Q^*. Typically this is the sequence Q^* that maximizes the joint likelihood $\Pr(\mathbf{O}, Q|\lambda)$.

The basic problem in HMM speech recognition is to find the most probable state path $Q^* = \{q_1^*, q_2^*, \ldots, q_T^*\}$, given a sequence of T observations $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T\}$ and a particular hidden Markov model λ. Thus Q^* maximizes in some sense the joint probability $\Pr(\mathbf{O}, Q|\lambda)$. Assuming a fixed HMM λ, this may be written

$$\Pr(\mathbf{O}, Q) = \Pr(\mathbf{O}|Q)\Pr(Q) \tag{3.3}$$

which rather neatly separates the recognition problem into two tasks: the acoustic modeling problem (finding $\Pr(\mathbf{O}|Q)$), and the language modeling problem of finding the most likely *a priori* state sequence $\Pr(Q)$. Language modeling is a difficult problem in its own right, and will not be considered here. Typically, individual hidden Markov models are sub-phone events, phones, or entire words; each model has a number of distinct states. The most probable state sequence Q^* identifies the most probable model sequence, which then may be considered the recognized string. For the experiments presented in Chapter 5, all (word) models will be considered equally likely, thus finding Q^* need only involve maximizing over $\Pr(Q|\mathbf{O}, \lambda)$.

Before any training or recognition can be done, a method must be found of determining the probability that an acoustic observation (event) $\mathbf{o}_t$ was emitted by a particular state s_i. The observations $\mathbf{O}$ are vectors from an underlying feature space that is generally continuous and high dimensional. (Typical feature spaces contain linear or mel-scaled cepstral coefficients, frame energy, LPC residual, and time differences thereof [53].) The output-probability model B consists of a set of S probability density functions (pdfs) on the feature space, one for each state in the model. Thus, the set of output probabilities maps the observation feature space to a set of N state-conditional probabilities, each of which reflects the probability distribution of observations emitted by a particular HMM state.

Finding a good model for B is critical to the performance of of the speech recognition system [20]. In practice, these pdfs can only be estimated from a limited set of training vectors, so a good model must be flexible enough to match the actual distributions yet constrained enough to generalize well to data not seen in training. In low-dimensional cases, an unknown pdf may be determined from the available data by a variety of methods. If the pdfs are of a known parametric form—e.g. a mixture of normal distributions—then the task is reduced to a well-known parameter estimation problem [31]. If no assumptions are made about the form of the distributions, there exist nonparametric pdf estimators such as K-nearest-neighbor and kernel estimators that may be used [31].

Feature sets used in speech recognition are high-dimensional, and therefore discrete or nonparametric pdf estimates can be extremely crude even for a reasonably large number of data points. As an example, consider a relatively modest 10-dimensional feature space. If each dimension is quantized even as roughly as, say, 10 equiprobable bins, it will take at least 10^{10} data points just to insure that every bin might have one point in it. Reasonable recognition performance depends on circumventing this "curse of dimensionality" [10], and finding a way to obtain good pdf estimates from a finite amount of training data.

3.1 Discrete models

In practice, output-probability models are conveniently divided into discrete and continuous flavors. Discrete models first map the feature space into a finite set of symbols or labels; each HMM state then has an associated stochastic vector of label output probabilities. Typically, feature vectors are mapped into the labels by finding the nearest vector from a fixed set of R reference vectors. This mapping from a continuous space to a finite, discrete set is termed

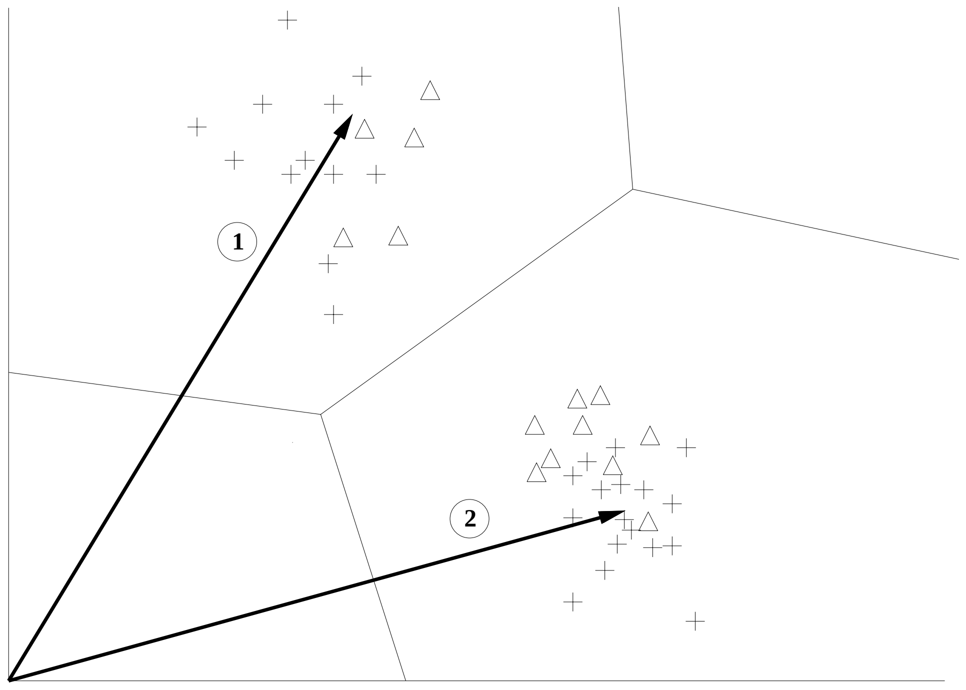

Figure 3.2: Vector Quantization.

"vector quantization," abbreviated VQ [59, 70], and is shown schematically in Figure 3.2. The data points (symbolized by $+$ and $\triangle$) fall into two major clusters, the centroids of which serve as reference vectors (symbolized by the labeled arrows). Any data point falling in the polygon surrounding a reference vector is assumed to belong to that cluster and is labeled with the corresponding reference vector code. This scheme is computationally attractive, and is widely used for speech coding as well as recognition. Unfortunately, this often results an comparatively poor acoustic model, for the following reasons.

- Vector quantization partitions the high-dimensional feature space into a comparatively small number of regions. All observation vectors falling into the same region are lumped together, and any discriminatory information between them is lost. Because the number of regions is relatively few and the dimensionality is high, the discretization of the feature space is necessarily crude.

- The discrete regions in feature space have fixed boundaries and are not robust to either bias or noise. Slight changes in the actual distribution caused by talker variation, recording environment, or noise can result in completely different reference vector selection, with disastrous consequences for recognition performance.

- Reference vectors, and hence the label regions, are typically found using an unsupervised algorithm such as K-means clustering [59, 31]. Such regions correspond to clusters in the overall data space and not necessarily to meaningful acoustic events.

- The very concept of "distance" is problematical, as a distance measure implies some constraint on the form of the distribution—for example, in K-means clustering, vectors that are "distant" in a Euclidian or Mahalonobis sense are considered "different" even though they may be semantically identical. Ideally, features should be quantized to minimize *recognition error* rather than some distortion metric [64].

3.2 Continuous models

In an attempt to overcome some of these drawbacks, probabilities may be estimated on the D-dimensional observation space $\Re^D$ rather than a discrete mapping thereof. This is typically done by training a set of M multidimensional Gaussian functionals to fit the observed feature data, illustrated schematically in Figure 3.3. (In the figure, the symbols $+$ and $\triangle$ represent data points as before, while the ellipses represent contours of constant probability for a single mixture term. Two mixture terms are depicted as modeling the two clusters.) The desired state-conditional probability may then be calculated as a weighted

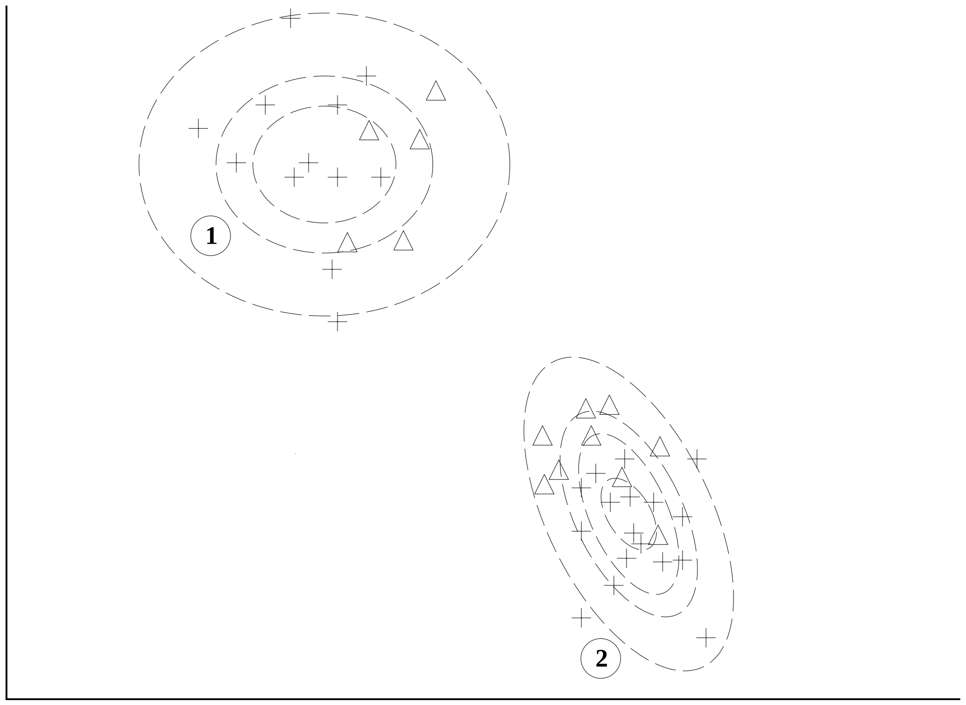

Figure 3.3: Continuous Mixture Density.

sum of multivariate normal distributions ($\mathcal{N}(\cdot)$):

$$b_j(\mathbf{o}_t) = \sum_{i=1}^{M} c_{ij} \mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{ij}, \boldsymbol{\Sigma}_{ij}) \tag{3.4}$$

where $\boldsymbol{\mu}_{ij}$ and $\boldsymbol{\Sigma}_{ij}$ are the corresponding mixture mean and covariance [72, 9]. The mixture coefficient c_{ij} is the relative contribution of mixture i to state j. Mixture coefficients are subject to the normalization condition that

$$\sum_{i=1}^{M} c_{ij} = 1. \tag{3.5}$$

Because this continuous model normally results in better recognition performance, it is generally agreed to be a better representation of the underlying probability space. A common variation on this scheme is known as "tied mixtures" or a "semicontinuous" model:

here a fixed set of M Gaussians are shared by all states in the model [67, 40].

$$b_j(\mathbf{o}_t) = \sum_{i=1}^{M} c_i \mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i). \tag{3.6}$$

This is advantageous because it reduces the number of parameters that must be estimated from finite training data.

A drawback of continuous models is that training the mixture parameters is computationally intensive, and subject to dynamic range problems [83]. As a parametric representation, many critical parameters (e.g. mixture size) must be determined *a priori*, independent of the observed data. Although a Gaussian distribution of speech data is plausible for no other reason than the Central Limit Theorem, there is little evidence that a normal distribution fits the observed data, especially when covariance matrices are assumed diagonal, as is usually the case. (See [12] for a gallery of scatter plots of cepstral and energy features. Many distributions look distinctly un-Gaussian, at least when projected onto two dimensions.)

3.2.1 Neural-Network-Based Probability Models

An increasingly popular output-probability modeling technique is the use of Artificial Neural Nets or ANNs. The huge variety in network types, philosophies, and training methods makes a concise summary impossible; the interested reader is referred to the canonical review article by Lippman [60]. Although substantial work has been done on purely network-based speech recognition [19, 86], it is generally agreed that a neural network structure is not well-suited to time-dependent data such as speech and therefore the most promising direction is the integration of ANN-based probability models with HMMs. To this end, work has been done using ANNs to estimate HMM state output probabilities, thus combining the best aspects of ANNs (powerful function estimation) and HMMs (time-dependent modeling and computational tractability) [15, 77, 78, 16].

3.3 Tree-based Probability Models

The conventional discrete and continuous acoustic models already described assume a parametric form on the densities involved. If there is a mismatch between the model form and the underlying densities, maximizing $\Pr(Q|\mathbf{O}, \lambda)$ will not necessarily lead to better recognition performance [20]. A discrete, non-parametric alternative to the above methods is proposed here. In this method, the feature space is partitioned by a decision tree into contiguous, non-overlapping regions called leaves (as discussed in Chapter 2). Associated with each leaf l_j is a set of probabilities $p_j(i)$, that a state s_i produced a vector landing in leaf l_j. These probabilities are used directly in the HMM recognition algorithm. Figure 3.4 shows schematically how the the tree-based model contrasts with continuous and VQ-based discrete models.

- The VQ probability model tessellates the feature space into a number of Voronoi polygons (the locus of points closer to the associated reference vector than any other vector). Reference vectors are commonly found using unsupervised methods and are independent of class distributions.

- Continuous models such as mixture density estimates fit Gaussian or other smooth basis functions to the observed training data.

- Tree-based models partition the feature space into boxlike leaf cells. Though obviously not as flexible as Voronoi polygons, it is practical to use two or three orders of magnitude more leaves than polygons (reference vectors) and hence the tree partition

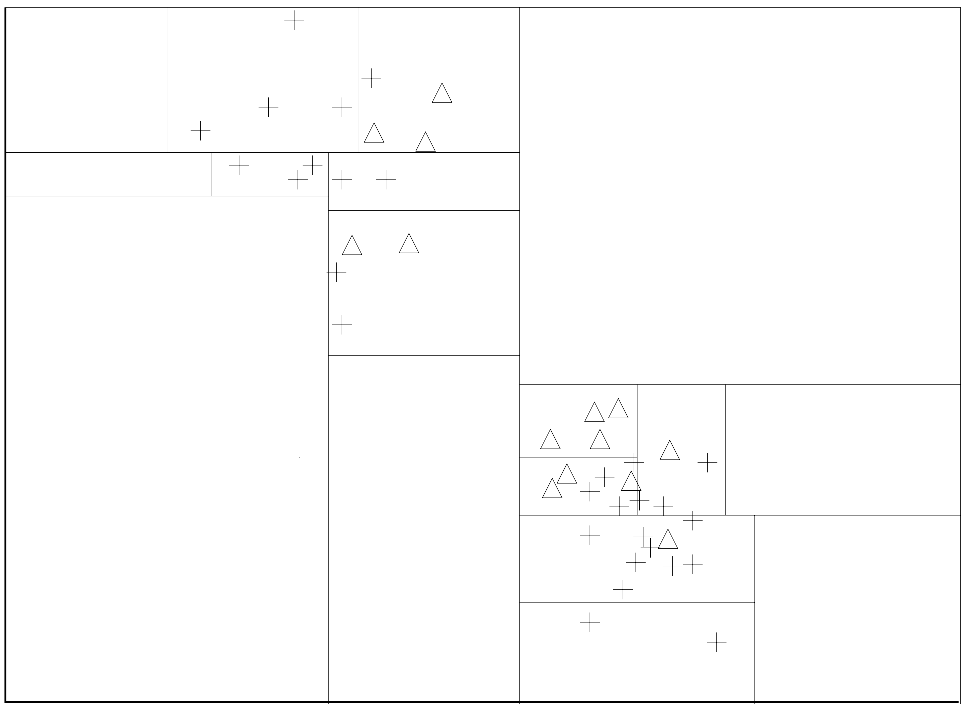

Figure 3.4: Tree quantization.

can exceed the spatial resolution of a VQ tessellation.

This acoustic modeling strategy has a number of interesting advantages over conventional approaches:

- Trees handle high-dimensional spaces gracefully, essentially by ignoring dimensions that don't aid the classification task. This allows the use of high-dimensional feature spaces that other methods may find problematic. For example, dynamic variation may be modeled by concatenating time-adjacent input vectors, which has shown to be an important way of circumventing the HMM independent-output assumption and adding context-dependent information.
- Because of the hierarchical nature, finding a tree-based output probability given the input vector is extremely fast. Classification speed means practical values of L on the order of several thousand or tens of thousands, where L is the number of discrete

regions in the feature space. Conventional VQ systems are limited to L of a few hundred by the costs of finding the reference vectors. Thus tree-based modeling allows high-resolution, non-parametric modeling of the underlying pdfs with the speed advantages of a discrete model.

- Trees can cope with categorical as well as continuous data, which permits the use of novel acoustic features. In addition, the tree-determined class label of the previous vector may be fed back into the tree as a feature, in a manner similar to a "recurrent neural network."

- Given a decision tree, new probability estimates may be found using Viterbi-labeled data. Such probabilities may be derived in linear time, rather than the iterative training required by parametric pdf estimators or artificial neural nets. This allows a practical method of speaker adaptation by reestimating probabilities as new talker data becomes available. This has potential both as a talker-adaptation method but also for speaker-independent tasks by using a small number of reference talkers and "adapting" between them.

- Decision trees are non-parametric and make no assumption on the form of the actual data distributions. Traditional VQ systems require the number of reference vectors, and hence the resolution of the feature-space quantization, to be set *a priori*. In a tree-based systems, the resolution (and hence the number of free parameters in the probability model) may be naturally adjusted by pruning the tree. This can be used to determine the appropriate tree size for, and hence the number of parameters to be estimated from, a given amount of training data.

- Once the feature space has been discretized by the tree, the relative importance of the individual feature dimensions may be discerned from the tree structure. This allows discrimination between feature sets, to find the features that best represent the underlying speech information.

The above discussion is not to suggest that tree-based models are a "magic bullet;" they are not. Drawbacks of the tree method include the following:

- The set of class boundaries is relatively inelegant: for the applications presented later, boundaries are constrained to be hyperplanes and normal to feature axes. This is less flexible than the Voronoi polygons in a nearest-neighbor VQ system, but can be compensated for by increasing the tree size (roughly analogous to increasing the quantization resolution). A complicated boundary may be fitted arbitrarily closely by increasing the number of leaves.

- A decision tree is still essentially a vector quantizer and suffers from most of the disadvantages of Section 3.1.

- A decision tree model is non-parametric and has many more free parameters than a parametric model of similar power, and will require consequently more storage. Because these parameters must be estimated from training data, a large amount of training data is required for good parameter estimates.

Chapter 2 discusses the construction and properties of decision trees, while Chapter 4 shows their use in the context of a HMM speech recognition. Experimental results are presented in Chapter 5.

This Tree is not as we are told, a Tree
Of danger tasted, nor to evil unknown
Op'ning the way, but of Divine effect
To open Eyes...

Milton

Chapter 4

Tree-based Probability Models

This chapter discusses how decision trees may be used to model the state-conditional output probabilities in an HMM speech recognition system. This is a two-part process consisting of tree construction, which may be done using the methods of Chapter 2, and output probability estimation. Once the tree has been built, it is used essentially as a vector quantizer, as depicted in Figure 4.1. The necessary output probabilities may then be estimated using the methods of Section 4.3.

4.1 Labeling Speech Data

Before the tree can be constructed, the speech data must be labeled, that is, all observations in the training set must be associated with some class label. (This is a necessary requirement of all "supervised" classification schemes such as Learning Vector Quantizers (LVQs), linear-discriminant analysis, and ANNs, as well as trees.) For speech, the very concept of "class" is not well defined, and depends to a large extent on the recognition task. For instance, in a large-vocabulary system, the appropriate class labels might be phones or sub-phonetic units, while for a small-vocabulary task they will be words or even phrases. Even given accepted

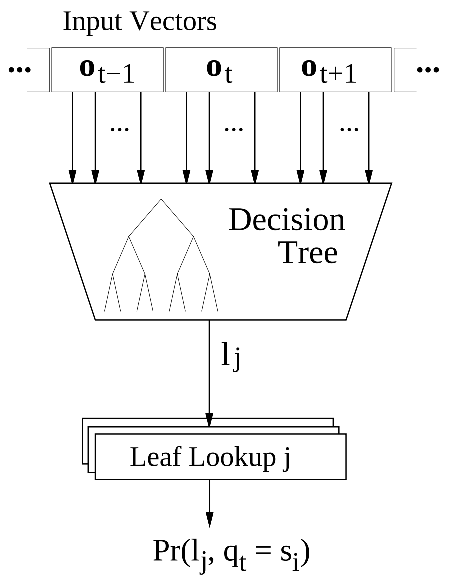

Figure 4.1: Contextual input to decision tree.

definitions of words, coarticulation effects will tend to blur the boundaries between them, so there may be no obviously "correct" segmentation. Labeling speech data is therefore a non-trivial undertaking. While there exist speech databases that have been labeled by experts[1], a commonly-used procedure is called Viterbi labeling, where a HMM-based recognizer is run on unlabeled data using a model constrained to the *a priori* known word sequence. Because of this constraint, much of the uncertainty (and hence error) is removed from the recognition process. The most probable state path may then be determined, which gives a maximum-likelihood alignment between model states and the speech signal. Because hand-labeling speech is onerous and error-prone, Viterbi-labeling techniques are widely used in practice [55, 62]. (Automatic labeling has the added advantage that even if is wrong, it is more consistently wrong than data labeled by humans.)

Viterbi-labeling speech data results in a sequence of feature vectors that have been labeled with the HMM states that most likely produced them. That is, each observation $\mathbf{o}_t$

[1]Actually, the speech data is usually segmented automatically; human experts review the segmentation and adjust it if necessary. [92]

is assigned a corresponding state $s_i, i \in \{1, 2, \ldots, S\}$, which serves as the class label c_i of Chapter 2. While in a word-based HMM system there is no guarantee that an HMM state corresponds to a meaningful, or even stationary, segmentation, the technique works well in practice and is widely used for supervised classifier training [76, 7]. Given the labeled data and the tree construction techniques of Chapter 2, it is straightforward to build a decision tree using Viterbi-labeled speech data.

4.2 Tree Probability Models for Speech

A tree $\mathcal{T}$ contains a number of terminal leaves $\mathcal{L} \in \mathcal{T}$. The set of leaves $\mathcal{L}$ partitions the feature space into L adjacent, non-overlapping regions, called *leaf cells* or *cells*. Let $p_j(i)$ denote the probability that an observation $\mathbf{o}_t$ emitted by state s_i falls in leaf cell l_j (where i indicates the particular state and j the particular leaf):

$$p_j(i) = \Pr(l_j | s_i). \tag{4.1}$$

A tree density model therefore consists of a set P of probabilities $p_j(i)$, which can be considered as S vectors of length L, where S is the number of states in the model and L is the number of leaves in the tree ($L = |\mathcal{L}|$). The number of free parameters is thus $S \times L$. The size of the tree may be chosen at will, allowing flexibility in the number of free parameters. If training data is limited, a small tree may be preferable because larger trees have too many free parameters to estimate reliably. Conversely, a large tree can yield a very detailed pdf model if there exists sufficient data to train it properly. A tree-based HMM model can then be denoted as $\lambda \overset{\triangle}{=} \lambda(\boldsymbol{\pi}, A, P)$, where P takes the place of the B matrix of conventional models. Note that if a tree is used as a vector quantizer, a model trained with quantized observations will have probabilities B identical to a tree-based model P. The difference is that in a conventional model, each *state* stores *leaf* probabilities while in the tree-based model, each *leaf* stores *state* probabilities with the following relation:

$$b_i(j) = p_j(i). \tag{4.2}$$

4.3 Estimating Tree PDFs

There are two general schemes for training a HMM model; the so-called "Viterbi training" and the Baum-Welch algorithm. Both of these may be used to estimate the tree probability model P.

4.3.1 Baum-Welch Estimation of Tree PDFs

A popular and computationally practical method of estimating HMM parameters is the Baum-Welch algorithm. This is sometimes called the "forward-backward" algorithm as it consists of two phases: computing the "forward" and "backward" probabilities $\alpha_i(t)$and $\beta_i(t)$ respectively. The forward probability $\alpha_i(t)$ is defined as the likelihood that an observation sequence up to time t ended in state s_i, given the model:

$$\alpha_i(t) \overset{\triangle}{=} \Pr(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t, q_t = s_i, |\lambda). \tag{4.3}$$

Similarly, the backward probability $\beta_i(t)$ is the probability that, starting in state s_i, an observation sequence runs from time t to the end (time T):

$$\beta_i(t) \overset{\triangle}{=} \Pr(\mathbf{o}_{t+1}, \mathbf{o}_{t+2}, \ldots, \mathbf{o}_T | q_t = s_i, \lambda) \tag{4.4}$$

in the notation of [73].

The forward-backward algorithm uses an efficient dynamic programming method to calculate (4.3) and (4.4). Once calculated, the probability of being in state s_i at time t given the model and observed data is denoted $\gamma_t(i)$, and may be found as follows:

$$\gamma_t(i) = \Pr(q_t = s_i | \mathbf{o}_1, \mathbf{o}_2 \ldots, \mathbf{o}_T, \lambda) \tag{4.5}$$

$$= \frac{\alpha_i(t)\beta_i(t)}{\sum_{i=1}^{N} \alpha_i(t)\beta_i(t)} \tag{4.6}$$

Given $\gamma_t(i)$ as above, the tree densities P may be estimated exactly as the output probabilities B of a conventional HMM:

$$p_j(i) = \Pr(l_j | q_t = s_i) = \frac{\sum_{t:\mathcal{T}(\mathbf{o}_t)=l_j} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.7}$$

where $\mathcal{T}(\mathbf{o}_t)$ denotes the terminal leaf "reached" by the observation $\mathbf{o}_t$ when classified by the tree $\mathcal{T}$. (Equivalently, $\mathcal{T}(\mathbf{o}_t)$ is the volume associated with leaf l_j that contains vector $\mathbf{o}_t$.) Equation 4.7 is just the expected value of $\gamma_t(i)$ for the observations that happen to fall into leaf l_j. This result means that the tree densities P can be iteratively re-estimated with the other model parameters in the Baum-Welch training algorithm. It should be noted that there is nothing novel in this approach; it is identical to Baum-Welch training for a conventional discrete HMM where the tree is used as a vector quantizer.

4.3.2 Viterbi Training

Viterbi training, also called *Viterbi extraction* [49] uses an existing HMM system to Viterbi-label a training utterance. This is done by constructing a special HMM constrained to follow the *a priori* known model sequence; only the correct word transitions are allowed. The

Viterbi algorithm then finds the maximum-likelihood alignment between the states and the acoustic observations. Once the state sequence is known, transition and output probabilities may be re-estimated by taking expectations exactly as in Baum-Welch training (section 4.3.1). Viterbi training is analogous to Baum-Welch training where the probabilities $\gamma_t(i)$ of being in state s_i at time t are constrained to be one (for the best-path state at time t) or zero (for all other states). The re-estimated model parameters can be used to find a new Viterbi alignment, and the process iterated; this is one form of the *segmental K-means algorithm*, which has been shown to converge to a local maximum [51, 71].

Given Viterbi-aligned data and a classification tree, it is straightforward to count S_{ij}, the number of data points from state s_i (i.e. the number of observation vectors aligned with state s_i) that wind up in leaf l_j. From this, the joint probability of the leaf l_j and state s_i may be estimated by dividing by the total number of data points N:

$$\Pr(l_j, s_i) \approx \frac{S_{ij}}{N}. \tag{4.8}$$

Any desired marginal or conditional probability can then be derived from the joint probability. For example, the conditional probability of the leaf node given the state is the joint probability divided by the state marginal:

$$\begin{aligned} p_j(i) &= \Pr(l_j|s_i) && (4.9) \\ &= \frac{\Pr(l_j, s_i)}{\Pr(s_i)} && (4.10) \\ &= \frac{\Pr(l_j, s_i)}{\sum_{l=1}^{L} \Pr(l_j, s_i)} && (4.11) \end{aligned}$$

It is thus straightforward (not to mention quite rapid) to estimate the tree output probabilities $p_j(i)$. The training data can then be relabeled using the new tree model, and the probabilities re-estimated. As before, this is completely analogous to Viterbi training a discrete HMM model where the tree is used as a vector quantizer.

4.4 Context Modeling

It is reasonably certain that the independent-output assumption is inaccurate for the speech signal, i.e., the observation likelihoods are not conditionally independent of time-adjacent observations. This assumption is made primarily for convenience; more complicated dependence models can be constructed at the cost of increased computation and storage requirements, and a substantial increase in training complexity [88]. Typically, the independent-output assumption is kept, and secondary features are added that indicate the time rate-of-change of primary features [90, 53]. These invariably improve recognition performance, albeit by increasing the feature space dimensionality.

One way of naturally adding contextual information is to concatenate time-adjacent vectors, yielding a higher-dimension feature space [20]. (This approach is also used in many ANN-based acoustic models [86, 16, 3].) For example, two adjacent n-vectors may be concatenated to form a single vector with $2n$ elements. This explicitly introduces time dependence into the feature space. Consider, for example, a one-dimensional discrete-time random process X having value x_n at time n. Imagine the two-dimensional space formed by time-adjacent values, i.e. plot x_n versus x_{n-1}. This is linearly separable into a positive-slope region and a negative-slope region by the line $x_n = x_{n-1}$. More complicated time dependencies can be modeled with more complicated boundaries. The drawback of vector concatenation is that the feature space dimensionality can get large enough to easily overwhelm conventional classifiers, even after dimensionality reduction by principal-component or linear-discriminant analysis [68, 20].

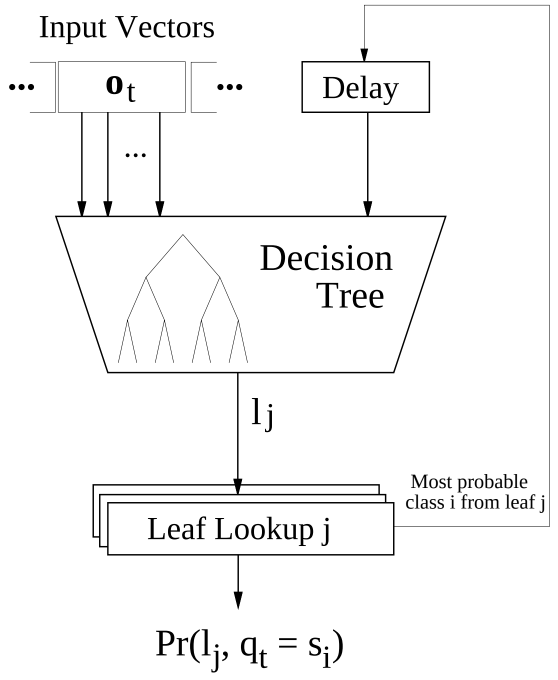

Figure 4.2: A time-recurrent decision tree.

Decision trees, on the other hand, can gracefully handle high-dimensional feature spaces and allow context modeling in a very natural way. Because of the supervised manner of tree construction, dimensions that do not yield information about the class domain are simply ignored, in contrast to other methods, where these dimensions must be explicitly considered. For example, in a mixture-based acoustic model the mean and covariance must be estimated for every dimension regardless of its importance in the classification task. Even in a computationally less intensive discrete system, a new input vector must be compared with a set of reference vectors using a distortion measure that must be computed on the entire dimensionality of the space.

4.4.1 Time-Recurrent Trees

Because trees easily handle discrete (categorical) features, a straightforward extension to a tree-based quantizer is to add the classification result of the previous vector to the feature space. The idea, similar to the use of "recurrence" in ANN-based acoustic models [78, 14, 16], is depicted in Figure 4.2. Here the most probable class of the last input vector is used, together with the current vector, as input to the decision tree. This can model time dependence without the large dimensionality increase of context concatenation. (The fact that feature values are continuous and the class label is discrete is of no consequence to the decision tree but could not be modeled with, say, a nearest-neighbor vector quantizer.)

While the actual leaf number itself could be used, analogous to the reference vector label in a VQ system, this makes tree construction problematical because the tree structure would need to be known before it was determined. In work done at Speech Systems Incorporated, a similar strategy was described that used a two-stage tree—the first stage selected leaf nodes based on the input vector or window, while the second stage selected leaf nodes based on a context window of first-stage outputs [2, 1].

An unexplored use of decision-tree acoustic models might be feature sets having categorical components, e.g. the binary output of a voiced/unvoiced speech detector (as suggested in [64]). Such features might be quite useful in discriminating stop consonants, yet would be awkward (if not impossible) to add to conventional classifiers.

4.5 Rocks and Hard Places: Undertraining and Overfitting

Acoustic models must necessarily be trained from a finite amount of data. In general, performance improves with additional training data, because it allows better estimates of the underlying probability space. To quote R. Mercer of IBM, "There's no data like mo' data" [6].

Every probability model has a large number of parameters which must be estimated from the training data. "Undertraining" refers to the situation where there is insufficient data to get robust estimates of the parameters. Probabilities will be “noisy” and the resultant model will perform poorly. On the other end of the spectrum is overfitting, where models essentially memorize the training data and are unable to generalize to new data—classification performance on training data will be excellent but will suffer on novel data. (Overfitting is often called “overtraining”; this is perhaps an unfortunate term because a good model will handle large amounts of data gracefully).

Nonparametric models are especially sensitive to overfitting, because they are so data-dependent. In the case of decision trees, overfitting is caused by trying to grow too large a tree from a given amount of data. Many leaves will have been grown just to fit clusters or configurations that occurred only by chance and that do not really represent the distribution from which the data was drawn. Because tree-based acoustic models can be grown arbitrarily large, it is important to determine when the trees are optimally sized. A mass-weighted split criterion automatically guards against overfitting because nodes with insufficient data will not be split. Perhaps the most widely-used method is to prune large trees according to classification results on an independent or cross-validation data set [18]. This is the approach used in the experiments section 5.6: the performance of the system on novel data is judged against different tree sizes.

4.6 Decision Trees for Speech Recognition

It is not surprising that decision trees have found widespread use in speech. Besides language modeling, investigators at IBM have been using decision trees to generate spelling-to-sound rules [5], and to model phonetic context. In a large-vocabulary speech recognition system, word models must be built from phone or sub-phone units, however, there is usually a number of different phone sequences that model a given word because of articulation or pronunciation differences (for example, "butter" is usually pronounced "budder," but "buT-Ter" is also correct and must be considered). This allophonic variation has been modeled at IBM by building a tree for each phone[2]. The trees are constructed by looking at the phonetic context of a large amount of training data; the leaves of the tree correspond to probable allophonic sequences and are used to construct an appropriate word model [6].

Interesting recent work at IBM applies similar phonetic context trees to the vector quantization stage [7]. Reasoning that the phonetic context influences the distribution of feature vectors, context-dependent trees are built for each class (phone). At the leaf of each tree, a diagonal-covariance Gaussian distribution models the observed vectors in that leaf. At recognition time, the models are evaluated across all leaves of all trees and the most probable tree model (class) is chosen as the VQ output. (Since there are over two hundred trees, each presumably having a large number of leaves, the efficiency of this method will be left for the reader to judge.)

A related use has been a hierarchical method for phoneme recognition. It is simpler to decide in which of two broad phonetic classes a given segment of speech belongs, rather than the precise phoneme. Such decisions can be made recursively until the classes consist of individual phonemes, in which case the speech segment has been classified. The resultant classifier is essentially a decision tree. An interesting twist on this approach uses neural nets to make the split decision at each node; this has been used for both phoneme classification [74] and Swedish vowel recognition [84].

Wightman and Ostendorf [89] have used decision trees to classify intonational features in speech. This is an excellent example of a decision tree's ability to handle both categorical data (e.g. syllable stress/unstress and final/not-final) as well as continuous features (pause

[2]Such a collection of trees is called a "wood."

time and mean pitch frequency).

Context Modeling

Practically all current speech recognition systems use some contextual modeling to circumvent the HMM independent output assumption [68, 80, 53]. Perhaps the most common method is to add features based on the rate of change of primary features, which implicitly include information from the nearby vectors used to compute the slope [90, 53]. Most network-based speech recognition systems model context more explicitly by using time-adjacent vectors (see, for example, [3, 14, 77, 19]). The Time-delay Neural Network (TDNN) approach introduced by Waibel et al. [86] not only uses 150 ms of context (15 adjacent frames) in the input layer, but also the context from a time-shifted hidden layer to further increase the "memory."

Tree-based Quantizers

The work presented here extends previous investigations of tree-based quantizers done at Speech Systems Incorporated (SSI) and by Ostendorf and Rohlicek [64].

Work done at SSI [2, 1] has used trees grown with the same MMI-based split criterion, though partitions were found through a gradient-descent search to find the optimal hyperplane split, rather than an exhaustive search of one-dimensional splits as done here. The input data considered a context window of three input vectors; acoustic features were identical to the *SPHYNX* system features: 12 cepstral and differenced-cepstral coefficients and energy/differenced energy at a 10 ms frame rate [53]. Once the basic tree was been constructed, a secondary tree was built to perform "segment" encoding. Time-contiguous runs of frames with the same leaf code (from the primary tree) were merged; a sliding window of this data, along with acoustic vectors, a rough phonetic classification, and duration information, is used to train the secondary tree to further segment the acoustic data. The segment-coded frames were then used as input to a simplified version of the *SPHYNX* HMM recognition system. Using the two-stage tree encoder resulted in a 33% error reduction and a net speedup of 1.6 over the VQ-based *SPHYNX* system. (The performance of a one-stage tree is not indicated; presumably the two-stage tree is necessary.) Problems were found using mutiple codebooks, and ultimately the performance was not found to be better than the continuous *SPHYNX* system [54].

Ostendorf and Rolicek discuss using the MMI criterion both for building decision trees and optimally reducing a set of VQ reference vectors. Their work using decision trees as a VQ did not perform better than a conventional VQ. This is probably because the trees used were relatively small (ca. 200 leaves), and did not take advantage of any more contextual information beyond frame differences. As discussed in Section 3.3, trees need to be large for a suitably detailed probability model, and the results presented in Chapter 5.6 show that adding contextual information improves performance substantially. They present interesting use of the MMI metric by using K-means clustering to find a large number of reference vectors, which they then "pruned" to a smaller subset by using an MMI metric. Thus reference vectors which did not enhance the classification capabilities were clustered with neighbors. This is a sensible and effective way to determine the proper number of reference vectors in a VQ system.

4.7 Analysis of Complexity

No discussion would be complete without an analysis of computational cost. For a discrete HMM, whether based on trees or nearest-neighbor VQ, this may be subdivided into the cost of training the classifier and the actual cost of classification. Because the classifier is trained "offline," it is less important that it not be burdensome. Classification, on the other hand, must be done in real time for a real-time recognizer, and so must not be overwhelming. The results of this section are summarized in Table 4.2.

4.7.1 Classifier Training Cost

Given N training vectors from a D-dimensional space, the complexity of the tree-growing algorithm may be analyzed as follows. (Cost parameters are conveniently summarized in 4.1, along with typical values.) For a given cell, a number of splits must be considered for every dimension. Constraining the partition to a single dimension, the number of possible splits is bounded above by one less than the number of points in the cell, because the cell could be split between any two data points with distinct projections on the split dimension (i.e. not sitting atop one another) [24]. This is typically an impractically large number of splits; in practice it serves to consider only a fixed number K of split points, especially because the MI split criterion is reasonably smooth. Splits must also be calculated for all D dimensions. Every possible threshold needs N_j scalar compare/count operations, where N_j is the number of data points inside the given cell volume. Thus each cell needs $N_j KD$ operations. The cells at a fixed level of the tree completely partition the feature space, hence every level will need NKD operations for all the cells at that level. If the tree has L leaves, it will have approximately $\log_2 L$ levels, and thus the total computational load is of order $NKD \log_2 L$ operations. For the systems discussed here, $\log_2 L$ is on the order of 10, D varies from 12 to 36, K is 40, and N is on the order of 200,000.

Reference vectors are typically determined by K-means clustering [59, 12]. This iterative procedure is time-intensive because at every data point must be compared with every refer-

Parameter	*Symbol*	*Typical value*
Utterance Length	T	1000
Feature Dimension	D	26
# Reference Vectors	R	256
# Codebooks	C	3–5
# Leaves in Tree	L	1024
# Training Vectors	N	200,000
# Model States	S	244
# Mixtures	M	256

Table 4.1: Summary of cost parameters.

ence vector to find the distance. Given R reference vectors, every iteration of the K-means algorithm requires R distance measures for every one of N data points to find the closest reference. In D–dimensional space a distance measure will take computation proportional to D for any useful metric. Thus an iteration of the K means algorithm requires order NRD computations; the total load is then $INRD$ over I iterations. For the LEMS system, typical numbers are $R = 256$, $D = 12$ and $N = 200,000$ as above.

For these numbers, the computational cost of tree-building and reference vector generation are roughly comparable. Because the number of reference vectors R in a VQ system is analogous to the number of leaves L in a tree quantizer, the advantage of the tree system is that computation grows with the logarithm of the number of regions rather than linearly. The computational advantage of trees becomes apparent when the number of leaves is substantially increased.

It should also be mentioned that both these methods require large amounts of training data, so memory access is perhaps as important a factor as computational speed. The KNN algorithm can access training data sequentially, so it may run faster (especially on an interleaved-memory machine) than a comparable tree generation program which accesses training data in an unpredictable manner.

4.7.2 Classification Cost

Tree construction and reference vector generation are done offline, and do not directly factor into the recognition or training speed. The actual classification cost is perhaps more important for real-time applications. A naive implementation of a minimum-distortion VQ system must compute a D-dimensional distance metric for each one of R reference vectors, for a cost of RD. There are ways of orchestrating the search in a hierarchical manner (essentially a tree) to reduce the number of necessary comparisons to $\log R$, but this doesn't reduce the cost of a comparison and may result in suboptimal performance because the minimum-distance reference vector is not always selected [75, 12]. Note that tree classification requires no multiplications (just comparisons), where a VQ classifier must compute a D-dimensional distortion metric. Probability estimation in both cases is a simple table lookup, once the vector has been classified.

For a continuous model, classification is replaced by probability estimation, which is more expensive. For each vector, the probability must be estimated for each state for both recognition and training. This requires the evaluation of M D–dimensional Gaussian functions, and weighting their contribution by the M mixture coefficients. Even assuming a diagonal covariance, this is not cheap. A further hindrance is that this computation must be done at run-time for a near-real-time recognizer. The precise computational penalty depends very much on the exact implementation and hardware [83].

4.7.3 Model Storage Requirements

The storage requirements for a discrete HMM Models are dominated by the output probabilities. A discrete HMM needs storage for SCR probabilities, where S is the number of states in the model, R the number of reference vectors, and C the number of codebooks. A

Operation	*Tree*	*VQ*	*Mixture*
Training	$\mathcal{O}(NKD\log L)$	$\mathcal{O}(INRD)$	
Classification	$\mathcal{O}(\log L)$	$\mathcal{O}(D\log R)$	—
Storage	$\mathcal{O}(SCR)$	SCL	$M(1+D+S)$

Table 4.2: Comparative costs of probability models.

tree-based acoustic model similarly needs to store SCL parameters. Because a tree-based model requires more leaves and codebooks for improved performance, it has more free parameters and consequently requires more storage. For the experiments presented in Chapter 5, $C = 5$ and $L = 1024$, versus $C = 3$ and $R = 256$ for the baseline discrete HMM system. Thus improved acoustic modeling comes with a cost of an order-of-magnitude more storage.

For continuous models, it will suffice to consider only the tied-mixture models which are currently in vogue. Assuming no codebooks are used to reduce the dimensionality, a tied-mixture model of the form of Equation 3.4 has M mixtures, which requires the storage of M means, and $M \times D(D-1)/2$ covariance parameters, and SM mixture coefficients. Typically covariances are assumed diagonal, so only MD covariance parameters are required. The total storage requirements are then $M(1 + S + D)$, or about 70,000 parameters. (This will increase roughly linearly with the number of codebooks.) Not surprisingly, a parametric model results in a much more compact representation.

Because function...
Function is the key.

Fugazi

Chapter 5

Experimental Results

This chapter presents some experimental results of using a tree-based probability model in the LEMS connected-alphadigit HMM speech recognition system [37].

5.1 The LEMS Speech Recognition System

Over the past several years, a talker-independent connected-alphadigit speech recognition system has been under development at the Laboratory for Engineering Man/Machine Systems at Brown University. While system details are well-documented elsewhere ([37, 33, 38, 34]), a brief overview is presented here.

The LEMS recognition system is designed for talker-independent, connected-speech alphadigit recognition. The recognition vocabulary consists of the alphabet ("*A*" through "*Z*"), the digits ("*zero*" through "*nine*"), two control words ("*space*" and "*period*"), and

Utterance	Pronunciation
Verne Hoover.	v-e-r-n-e-space-h-o-o-v-e-r-period
126370 exhibit	1-2-6-3-7-0-space-e-x-h-i-b-i-t
Suez. 6047088	s-u-e-z-period-space-6-0-4-7-0-8-8
3sh1ph1 coequal	3-s-h-1-p-h-1-space-c-o-e-q-u-a-l

Table 5.1: Typical utterances (after Hochberg [37]).

Data Set	Talkers	Men	Women	Utterances	Time (min)
TEST	20	12	8	321	26.7
(SMALL)	10	5	5	145	11.3
TRAIN	96	64	32	4398	369.5
(MALE)		64	0	2912	245.6
(FEMALE)		0	32	1486	123.9
Total	116	76	40	4719	396.2

Table 5.2: The LEMS Speech Database.

initial and final silences. Utterances consist of random digit sequences, random alphadigit sequences, and letters from dictionary words; a typical set of utterances is displayed in Table 5.1.

Speech is sampled at 48 kHz by a Sony PCM-2500 R-DAT recorder. It is then downsampled to 16 kHz and transmitted to a Sun workstation via a custom-built interface [33]. On the workstation, the speech data is segmented into 40 ms frames, Hamming-windowed, and processed into feature vectors. Each feature vector consists of 12 LPC derived cepstral coefficients, 12 delta-cepstral coefficients, normalized frame energy, and delta frame energy. Overlapping each frame by 30 ms results in a 10 ms feature vector rate. For the "baseline" discrete processing, each feature vector is then quantized by finding the nearest reference vector from a "codebook" of reference vectors. Three codebooks are used, for subsets of the feature space consisting of the cepstra, delta cepstra, and energy/delta energy features. For each codebook, 256 reference vectors were determined using a K-nearest-neighbor algorithm with a Mahalanobis distortion metric [12]. Thus each feature vector is labeled with codewords from each of the three codebooks, which are assumed independent. The quantized speech data is used to both train and test the HMM-based recognition system, described in Section 5.1.1.

HMM parameters are estimated from a training corpus of speech data. At LEMS , 116 talkers of American English each contributed about 3 minutes of speech, for a total of about 6.5 hours of data. This was arbitrarily divided into a 20 talker TEST set and a 96 talker TRAIN set. A "SMALL" subset of data from 10 arbitrary talkers in the TEST set was used to expedite testing of different tree and model configurations. Recognizer performance analysis was done solely on the TEST and SMALL data sets, which are completely independent from data in the TRAIN set used for model training. For the gender-dependent models of Section 5.7, the TRAIN set was subdivided into the MALE and FEMALE subsets, consisting respectively of data from all the male and female talkers in the TRAIN set[1]. Table 5.2 summarizes information about the data sets.

A drawback of conventional HMM systems is that the duration distribution in a given state is constrained to be exponential. It has been shown [39] that the exponential distribution is not a good model for the duration of many acoustic events which the HMM states are assumed to represent. Different duration distributions can be obtained by replacing the Markov chain with a semi-Markov chain and explicitly including duration in the HMM model. This representation—referred to as the continuously variable duration HMM (CVDHMM)—is described by Levinson et al. in [56] and its proponents have shown improved performance over traditional systems [61, 35, 36]. Though some find the computational cost of explicit duration modeling to be prohibitive, it has been used to good effect in the LEMS speech recognition system.

The recognized string is aligned with the correct string using a dynamic programming procedure [82]. Recognition error is defined as the ratio of inserted, deleted, and substituted

[1]Disclaimer: it was not the intention of the database designers to discriminate against women. The fact that women are represented in more than token numbers is due to yeoperson efforts made by the LEMS database recruiters.

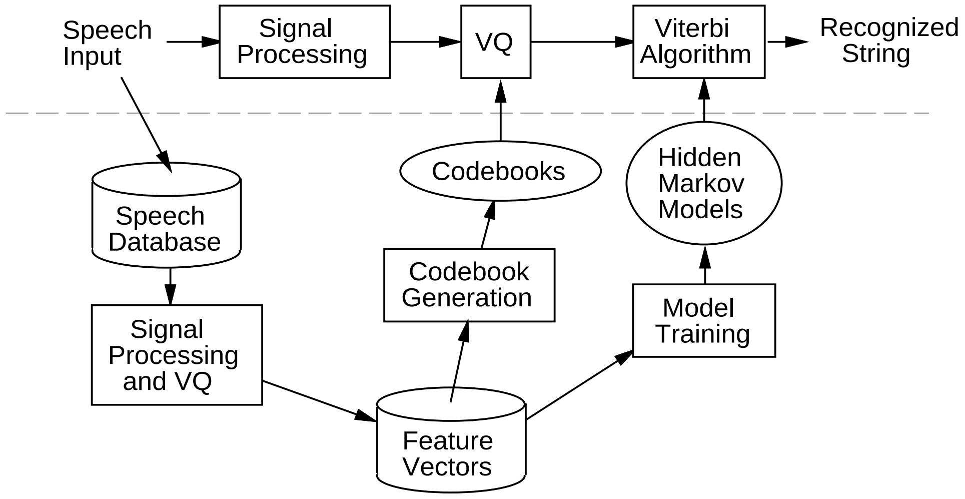

Figure 5.1: Speech recognition system block diagram.

words to the number of actual words in the utterance:

$$\text{Error} = \frac{\text{Substitutions} + \text{Deletions} + \text{Insertions}}{\text{Words}} \times 100\%. \tag{5.1}$$

5.1.1 The baseline HMM system

A "baseline" discrete HMM system was used both for a performance reference and to Viterbi-align data for tree construction and training, As previously mentioned, the system uses 3 codebooks of 256 reference vectors each, corresponding to the cepstral features, delta cepstral features, and energy/delta energy. State duration was modeled with an explicit Poisson distribution, and the models were trained on the TRAIN data set. Though a word-bigram language model is easily incorporated into the HMM structure, resulting in substantial performance gains, it was not used here: all word transitions are considered equally likely (a full-perplexity grammar). Under these conditions, the system achieved 84.3% performance on the TEST set and 83.92% performance on the SMALL test set. (The difference is due to talker variation.)

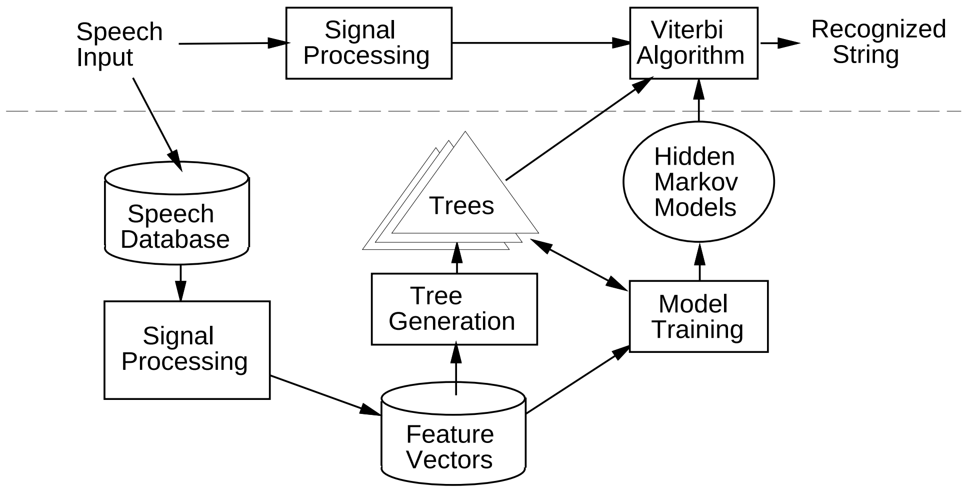

Figure 5.2: Tree-based HMM recognition system.

5.2 Tree-based probability models

The baseline LEMS connected alphadigit speech recognition system is shown schematically in Figure 5.1. Symbols below the dotted line represent "off-line" training programs and database files that support the real-time recognition procedure, shown above the line. In a tree-based HMM system, trees take the place of both the vector quantization procedure and the HMM output probabilities. Because quantization is so rapid, it may be integrated into the training/recognition procedures as described in Section 4.2. Alternatively, the tree may be used as a stand-alone vector quantizer. Because the the quantized feature symbols must then be stored, this is a computation/storage tradeoff. For the experiments of this chapter, probabilities were integrated into the tree structure with little penalty because of the classification speed of the tree.

5.2.1 Codebooks and Independence

A cepstral-domain representation of the speech signal is almost universally used for speech recognition. A generally recognized advantage of cepstral features is the individual dimensions are relatively uncorrelated. This is principally because the Fourier basis is orthogonal, though it has also been suggested that the Fourier basis is a good approximation of the Karhunen-Loéve expansion of spectral data (the expansion along principal-component axes) [79].

True dimensional independence is an especially useful property, because it greatly simplifies probability estimation. In a mixture-density system, the (full) covariance matrix of Equation 3.4 can be approximated by its main diagonal; assuming off-diagonal elements are zero saves considerable amounts of both computation and storage.

In a discrete system, the "curse of dimensionality" may be somewhat mitigated by using multiple feature sets, or "codebooks." Reference vectors and their corresponding probabilities are estimated for a lower-dimensional subspace of the feature space. Each subspace has a corresponding set of reference vectors, referred to as a "codebook." Joint probabilities on the high-dimensional feature space can be approximated by the product of lower-dimensional codebook probabilities: this has the advantage of increasing the number of effective quantization bins exponentially with the number of codebooks. (To see this, consider a number of one-dimensional, two-label codebooks. The first codebook divides the feature space into two regions; using the second gives four regions, corresponding to all 4 combinations of the two labels from each codebook. A third codebook yields 8 possible combinations, and so forth.)

The baseline LEMS system uses separate codebooks for cepstra, differenced cepstra, and energy/delta-energy. This is arguably the wrong approach because cepstral features are probably more correlated with time rather than dimension, that is, a given cepstral coefficient is more likely to be correlated with its time-adjacent values than other cepstra at the same time. The tree experiments of Section 5.6 investigate the use of context in the decision tree. As described in Section 4.4, this dramatically increases the feature space dimensionality. At the extreme, nine adjacent 15-dimensional vectors are concatenated, resulting in a feature space with 135 dimensions (!) In order not to get unduly sparse relative frequency counts, the "codebook" approach is used to reduce the dimensionality. One vector has twelve cepstral coefficients and energy, delta energy, and LPC residual; this is divided into 5 three-dimensional subspaces: cepstra 1–3, 4–6, 7–9, 10–12, and the energy/residual features. Trees are then constructed for each of the five subspaces; adding the (reduced) adjacent vectors increases the dimension to a maximum of 27, a much more manageable number. As mentioned above, these spaces can be considered reasonably independent; hence the probability estimate from a given observation may be computed by the product of the probabilities from each of the five trees[2]. Other feature space partitions are of course possible: the number of codebooks, the size of the trees, and amount of training data all probably interact to affect recognition performance.

5.3 Analysis of Dimensional Importance

A useful feature of a decision tree is that it can show each dimension's relative contribution to the decision task. This is demonstrated on real speech data in this Section. A tree was constructed using data from all talkers in the test set; the dimensional importance is charted in Figure 5.3. As might be expected, the energy and low-order cepstral dimensions contribute the most to the split criterion. It is interesting to note that the high-order

[2]Instead of assuming independence, Ostendorf and Rohlicek use the identity

$$I(X,Y;C) = I(X;C) + I(Y;C) - I(X;Y)$$

to construct trees from the multiple feature sets. Given an existing tree, a new tree may be constructed such that splits which yield only redundant information are inhibited. Thus splits are considered only when they add information not accounted for by splits in existing trees; this is said to make the tree outputs more truly independent [64].

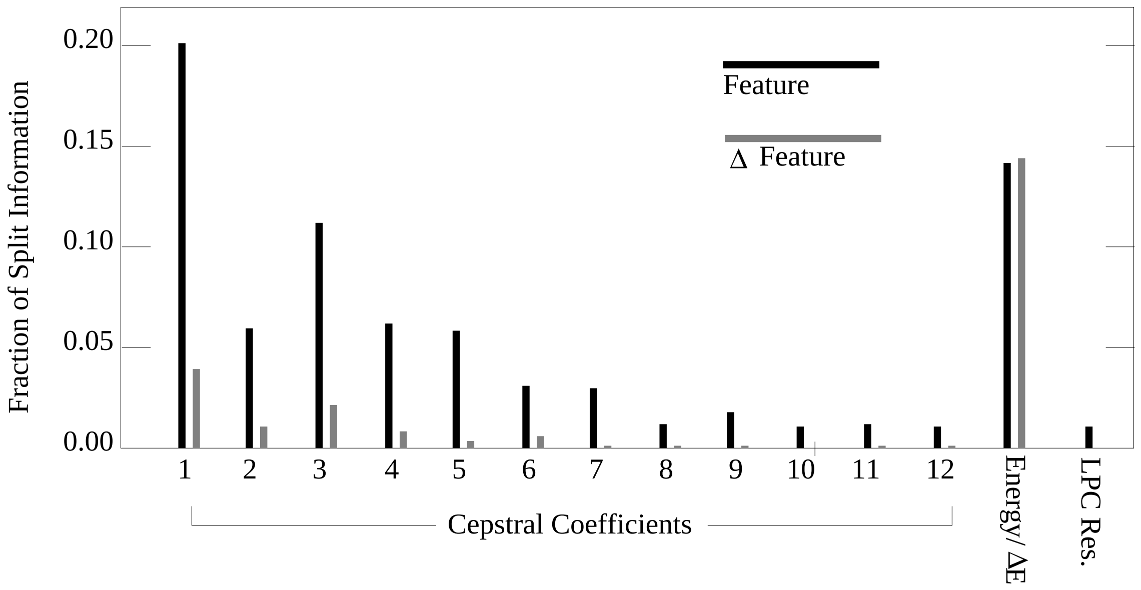

Figure 5.3: Split information of features.

differenced cepstra contribute little or nothing; this is perhaps due to the fact that high-order cepstra encode high-"quefrency" spectral features rather than the gross spectral shape. Thus high-order cepstra probably depend more on the vocal excitation rather than the vocal tract configuration, and therefore contain less information about the phonetic category of the data.

Figure 5.3 shows clearly the energy, delta energy, and low and mid-order cepstra are the most important features. These results agree reasonably well with those of Bocchieri and Wilpon [13] and Paliwal [65], where dimensions were ranked using a ratio of in-class variance to between-class variance. In Juang and Rabiner [52], performance recognition was improved by "liftering" the cepstra—in effect, emphasizing the mid-order cepstral coefficients at the expense of high and low-order cepstral coefficients. This conflicts with the results that show the low-order cepstral coefficients are quite important. This is perhaps because the LEMS database was recorded on a single microphone while the experiments of Juang et al. were done on telephone speech, presumably from different handsets. Because the first-

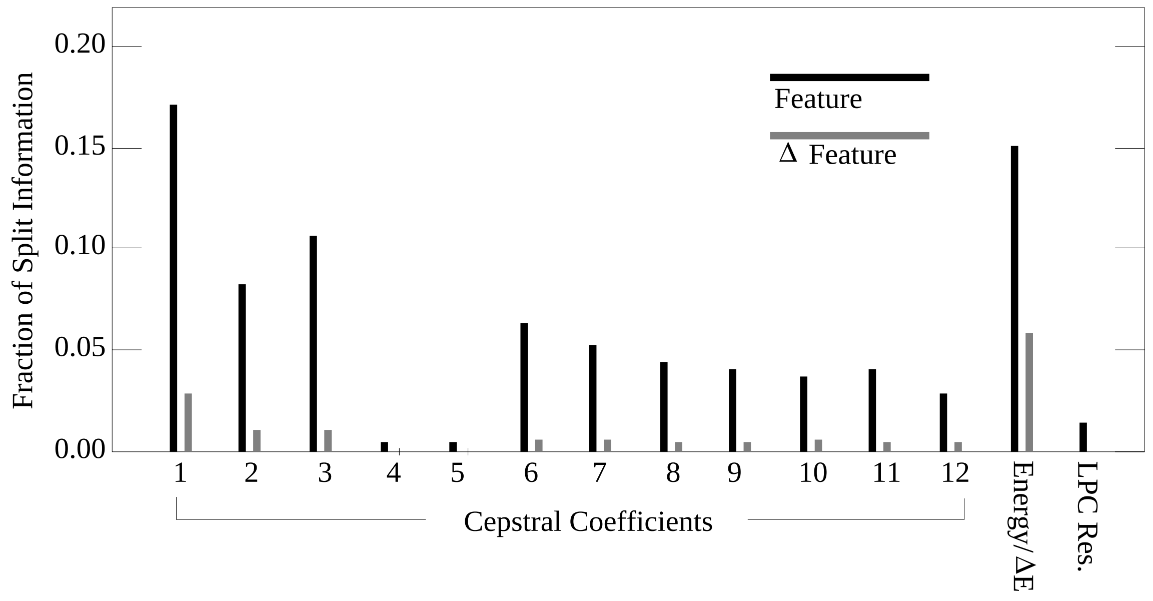

Figure 5.4: Split information of features where cepstral coefficients 4 and 5 have been replaced with uncorrelated noise.

order cepstral coefficient is a measure of spectral tilt, it will vary strongly with different microphones and recording environments, thus de-emphasizing it may tend to reduce the environmental effects on recognition performance.

As a test, a tree was built with identical data save that cepstral coefficients 4 and 5 were replaced with white Gaussian noise having similar means and variances. The resulting dimensional importances are shown in Figure 5.4. As expected, the relative importance of these features is negligible because the noise is completely uncorrelated with the segmentation. (The importance is not identically zero because random correlations can occur, especially in leaves with few data points.) This also shows an important lower bound on the "importance" axis—features having this much importance (or less) are essentially useless for the recognition task. A further test was to replace a feature dimension with "data" proportional to the vector label. Not surprisingly, all splits were performed on this dimension.

It has been shown ([90]) that the addition of second-differenced cepstra to the feature set can increase recognition performance. This, and the relative unimportance of the differenced

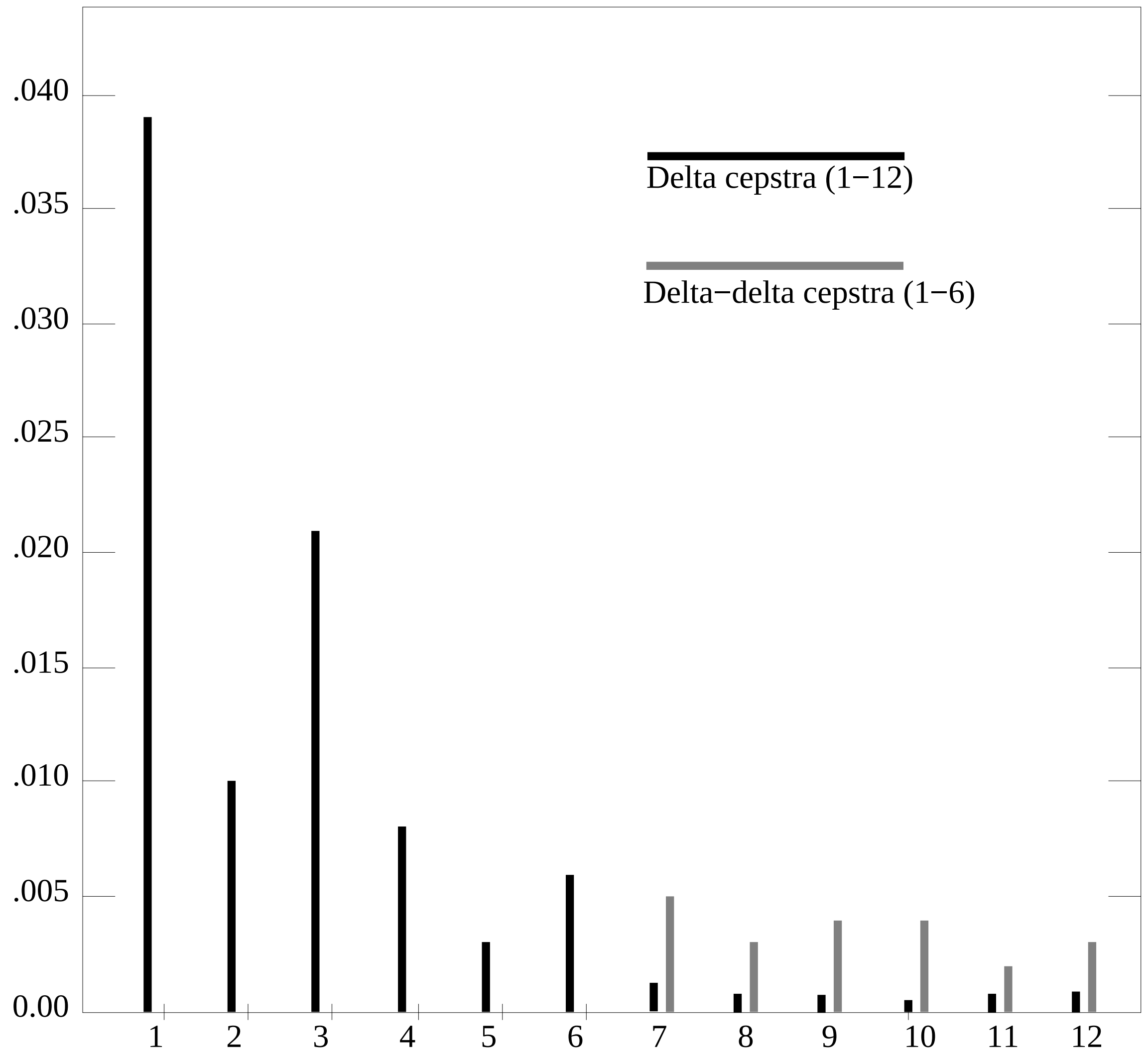

Figure 5.5: Split information of double-differenced cepstral coefficients.

high-order cepstra, suggested the following experiment: the six highest-order differenced cepstra were replaced with the second difference of the low-order cepstra. Figure 5.5 shows the relative importance of the low-order delta-delta cepstra with the importance of the first-order single-differenced cepstra they replaced. The results showed that the second-differenced features had three to four times the relative importance of the high-order delta cepstra, but the absolute amount was still relatively small, and probably not enough to significantly enhance recognition performance.

5.4 Explicit Context Modeling

Section 4.4 discusses the concatenation of adjacent feature vectors to explicitly model context, at the cost of increasing the feature space dimensionality. The ability to extract information from such high-dimensional spaces is a prime advantage of tree-based systems, and it has shown to substantially improve the accuracy of the tree-based recognition system. It is not necessary to concatenate strictly adjacent vectors; Figure 5.6 shows how vectors separated by a "stride" of r time ticks may be used to increase the context window—the length of time considered—without increasing the dimensionality. The dimensional-importance metric can be used to judge any feature dimension, including the dimensions added by concatenating features. Figure 5.7 shows the dimensional importance of concatenating five adjacent feature vectors. (This is found by summing the dimensional importance of each dimension across adjacent vectors—a "dimension marginal.") The five graphs show the effect of looking at 5 vectors an increasing distance apart, from adjacent (top graph) to a stride of 5 frames, or 50 ms (bottom graph). In the LEMS system, feature vectors are computed from 40 ms windows that overlap each other by 30 ms, or 75%. Thus each frame contains much the same information as its neighbors. This is evident from Figure 5.7, which

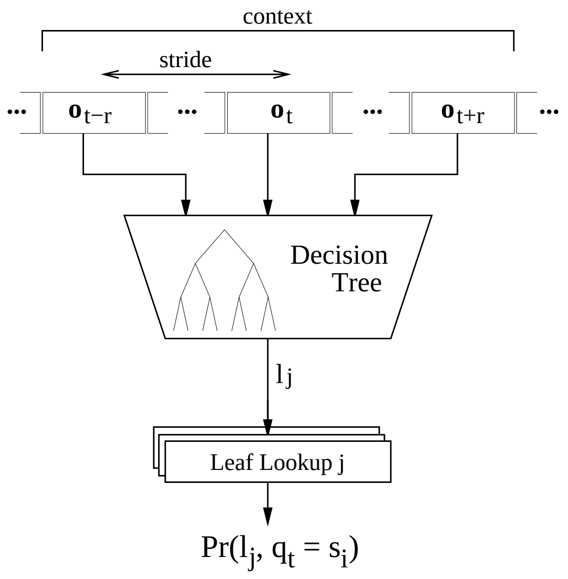

Figure 5.6: Context and stride for tree input.

shows that for a small stride value, most of the contextual importance comes from the vectors furthest away. This makes sense: vectors within one or two frames will be correlated because they are generated from much the same data. The more distant vectors will have more contextual information because they are less correlated. The "stride" variable was introduced primarily to overcome the overlap problem; it was hoped that increasing the stride value could incorporate information from reasonably time-distant vectors without a large dimensionality increase. Unfortunately, increasing the stride did not appreciably improve recognition performance.

5.4.1 Run-Length Adjustment

In Providence, the the probability of rain on any day is, say, 20%. The probability that Pat carries an umbrella is also near 20%. If the two events are assumed to be independent, the joint probability of the two events is the product of the individual probabilities, or 4%. In fact, the two events are probably well-correlated and the true joint probability (the chance that it is both raining *and* Pat carries an umbrella), is nearer 20%. Thus wrongly assuming

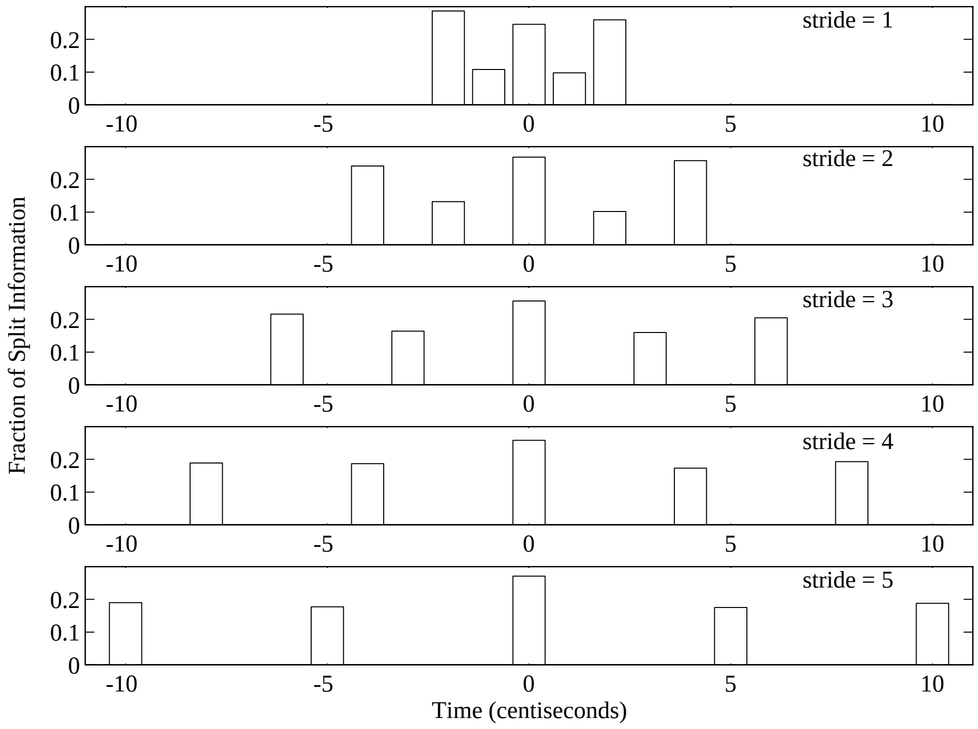

Figure 5.7: Context dependency: split information vs. stride.

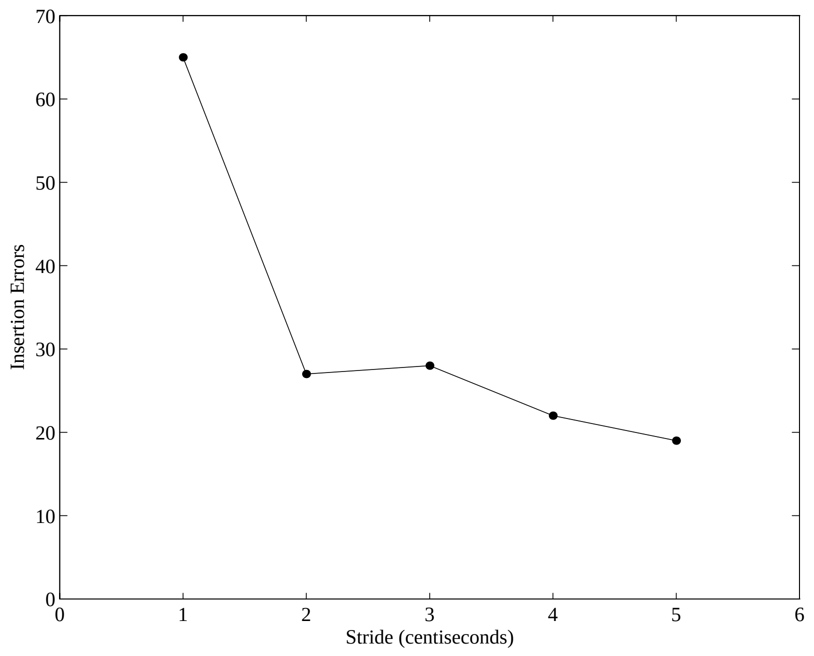

Figure 5.8: Insertion errors vs. stride.

independence can result in gross underestimates of true joint probabilities; the error gets worse as the independence assumption is increasingly violated.

An indication that contextual information improves the acoustic model is shown in Figure 5.8, which plots the number of insertion errors versus the stride value. It is a well-known (but little-discussed) fact that the HMM output probabilities must be adjusted to yield for the best recognition performance [20]. This is because the independent-output assumption underestimates the true (joint) probabilities of long-duration events when they are approximated by the product of independent outputs. Thus long-duration events such as vowels are considered less probable than they actually are. The net effect is to force an excessive number of insertion errors, as it becomes more probable to jump to a new word or state than to dawdle in an (erroneously) improbable long-duration event. In the LEMS system, this is fixed by exponentiating the word-to-word transition probabilities by a value greater than one. This is equivalent to Brown's "language model match factor," (LMMF) where observation-dependent probabilities are exponentiated by a factor (the LMMF) less than one [20]. The effect of the LMMF is to de-emphasize output probabilities in favor of word transition probabilities and therefore reduce insertion errors. This process, called "run-length adjustment," is used in many connected-speech recognition systems[53, 21]. The best LMMF value is determined empirically, typically by finding the value that roughly equalizes insertion and deletion errors.

Experiments presented in [39] show that a better state duration model requires a LMMF closer to one, (or equivalently, fewer insertion errors if the LMMF is unity). An acoustic model that better captures the time-dependence of the features should therefore have fewer insertion errors, given a LMMF of unity. This is exactly the result shown in Figure 5.8: as the stride is increased the number of insertions decrease. Especially significant is the large decrease when the stride is increased from one to two. With a stride of one, a context window of 5 means that the end vectors in the window will be well-correlated with the central vector (because they are computed from overlapping frames). Increasing the stride to two means that end vectors are separated by 40 ms and thus represent entirely novel data.

The LMMF may also be used to mask a variety of other sins. For example, assuming codebook independence will also cause output probabilities to be underestimated; this will have an effect similar to the independent-output assumption, and may be similarly fixed by adjusting the LMMF. The relative contributions of codebook and output dependence to the required LMMF is not known and perhaps deserves further investigation.

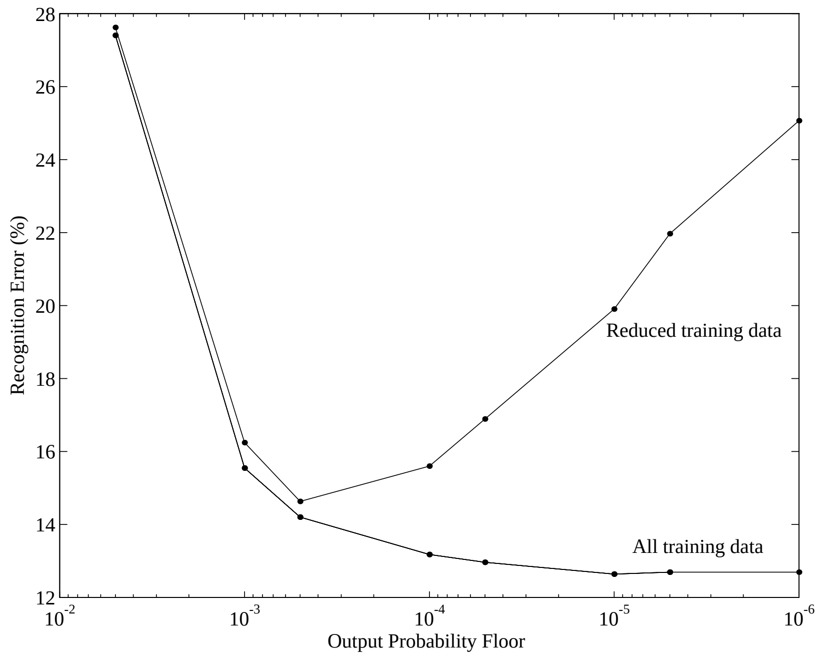

Figure 5.9: Recognition performance vs. probability floor.

5.5 Output Probability Smoothing

When training the leaf probabilities, some states will emit certain observations with low enough probability to be effectively zero for a finite set of training data. Consequently, some probabilities $\Pr(l_j|s_i)$ will be estimated as zero. This is undesirable for the following reasons:

1. Though certain observation emissions may not be seen in the training data, they may still occur with some small yet non-zero probability. Setting the probability of these occurrences to zero on the basis of the training data will cause problems when they occur in novel data.

2. In many systems, probability computations are done in the logarithmic domain both to simplify multiplication as well as to avoid underflow. Zero probabilities are awkward to represent in logarithmic form and are generally avoided [83].

Though this problem has more elegant solutions (such as "deleted interpolation"), a convenient if crude method is to set vanishing or zero output probabilities to some small yet non-zero "floor." That is, values of $p_j(i)$ less than some p_{floor} are set to p_{floor}, and renormalized if necessary.

$$p_j(i) = \begin{cases} p_j(i), & p_j(i) > p_{\text{floor}} \\ p_{\text{floor}}, & p_j(i) \leq p_{\text{floor}} \end{cases} \tag{5.2}$$

The effect of p_{floor} depends to a large extent on the amount of available training data. Figure 5.9 shows the effect of varying the floor value for both a large amount of training data (all data in the TRAIN set) and a small amount (approximately 10 % of the TRAIN set).

Essentially, p_{floor} is a guess about state emission probabilities not seen in the training data. If p_{floor} is too small, then the probabilities of unseen events are underestimated, and performance suffers. Conversely, if p_{floor} is too large, then then the probabilities of unseen events compete with high-probability events and performance again suffers; this is illustrated in the top curve of Figure 5.9. The best value of p_{floor} is determined heuristically by finding the value that maximizes recognition performance. Given sufficient training data, probability estimates will be accurate: increasing p_{floor} can only hurt performance by (wrongly) increasing the probability of unlikely events. The bottom curve of Figure 5.9 shows that performance does not deteriorate with decreasing p_{floor}; this is good evidence that there is sufficient training data to reliably estimate the tree probabilities.

5.6 Tree-based HMM recognition

Figure 5.10 shows the recognition performance versus tree size on the SMALL data set of Table 5.2. First, training data was Viterbi-aligned using the baseline HMM system

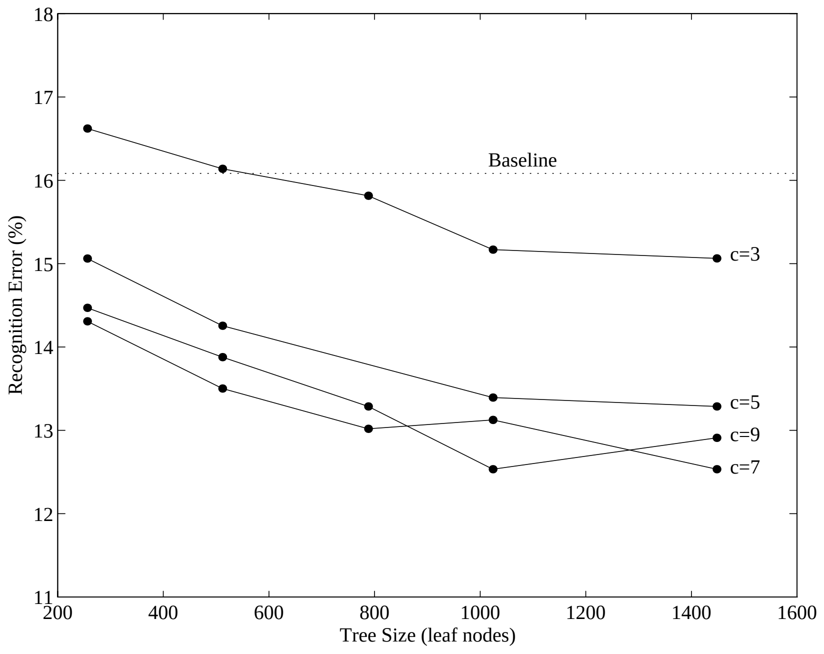

Figure 5.10: Recognition performance vs. tree size.

described in Section 5.1.1 (the performance of the baseline system is indicated by the dotted line). Vectors were labeled with the most probable state from the 244 total states. (Each word model was assigned from 4–12 states depending on the complexity of the word.) Note that states may not correspond to any "meaningful" sub-word segmentation; also the probabilities from many states may look quite similar, for example the later states of words in the E-set. For this experiment, trees were built using 4 utterances from each talker in the TRAIN set, or approximately 10% of the training data. Trees were built using context windows of 3, 5, 7, and 9 time-adjacent vectors; these correspond to the different curves in the figure and are labeled with the width of the context window. Once constructed, the trees are easily pruned to any desired size—in the experiment, trees were pruned to 256, 512, 1024, and 1448 leaves. Five codebooks were used, comprising cepstral coefficients 1–3, 4–6, 7–9, 10–12, and energy/delta energy. (Delta cepstral features were not used because the context should provide the equivalent information). Leaf probabilities were trained from all data in the TRAIN set, using relative frequency estimation (one iteration of the Viterbi update, Equation 4.9). To produce the recognition results shown in the figure, the transition and (Poisson) duration probabilities were unchanged from the baseline model and the output probabilities were obtained from the tree models. The results of Figure 5.10 show that including context substantially improves the recognition performance, reducing the recognition error by as much as 20%. Again note the improvement as the context window is increased from 3 to 5: this is the effect of including vectors generated from novel (non-overlapped) data. Increasing the context window much beyond 7, however, did not substantially improve recognition performance: beyond this distance, either significant contextual information was not captured by the tree model or else it is simply not there. Also, increasing tree size beyond a certain level does not appreciably improve recognition performance, even though the probability model is presumably more detailed.

The baseline performance is shown for reference only. The baseline system used only 256 reference vectors in three codebooks; given the 244 model states resulted in $3 \times 256 \times 244 = 187,392$ parameters in the output probability model. For comparison, the five-codebook, 1024-leaf tree model has 1,249,280 parameters, nearly an order of magnitude more.

5.7 Gender-Dependent Modeling

A large source of talker variation may be attributed to the pitch and vocal-tract differences between men and women. This is an example of speech signal variability that is *independent* of the information content: there is, of course, no substantial difference in the intelligibility of men and women talkers under normal conditions. This variation will, however, affect the performance of a recognition system, almost always adversely, because the natural variation

Tree Built	Tree Trained	Men	Women	Best
all	all	9.98%	17.70%	12.92%
all	MALE	9.68%	37.65%	11.64%
all	FEMALE	35.00%	14.84%	
MALE	MALE	9.02%	42.86%	12.48%
FEMALE	FEMALE	35.46%	17.67%	

Table 5.3: Recognition error of gender-dependent model experiments.

can be as large as or larger than the intrinsic differences in the various parts of speech. A recognition strategy which can minimize the unimportant differences should perform better.

One way of accomplishing this is to use different models to account for talker variation. This approach has been used in a number of recognition systems, for example at SRI [63], IBM [9] and in the *SPHYNX* system [53]. A simple but useful example is to construct different models for male and female talkers, by training each model on segregated data. This has a number of benefits: gender is a large yet discrete source of variation, and it is usually easy to separate talkers by gender.

As discussed in Chapter 4, constructing a tree-based probability model has two data-dependent steps: building the tree and training the leaf probabilities. These two steps may be done in a gender-dependent or -independent manner. The top line of table 5.3 shows the recognition error using a gender-independent model (this was a tree model using a context window of 7, pruned to 1448 leaves, identical to the model used in Section 5.6). The next two rows show the results of training the same tree on the MALE and FEMALE data sets to form two gender-dependent models. The next two lines show the effect of actually building trees with gender-dependent data. This did not result in improved recognition accuracy, perhaps because less data was used to build the trees. These results show that gender-dependent modeling improves recognition performance only slightly, and may not be worth the trouble. The surprise here is not how well the talkers did on the same-gender

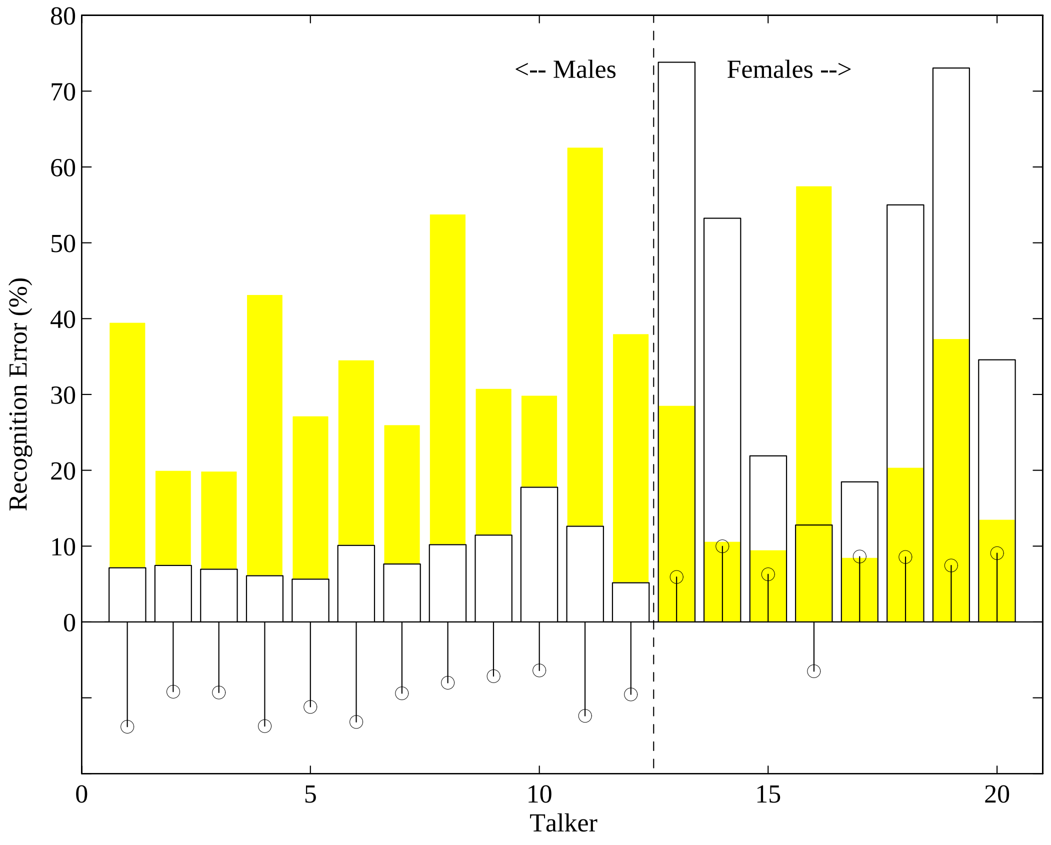

Figure 5.11: Individual talker performance on gender-dependent tree models. (Gray = female-trained model; white = male-trained model. Stems ($\dagger$) show classification error difference between female and male models [on an arbitrary scale].)

model, but how poorly talkers did on the incorrect model. These results also show that the data used for leaf-probability training is much more important than the data used to construct the actual tree. Not having to rebuild a tree is a definite advantage: Section 5.8 discusses making talker-dependent acoustic models by retraining the leaf probabilities of talker-independent trees.

Figure 5.11 shows the performance of individual talkers with the talker-independent tree trained with MALE data (white) and FEMALE data (gray). As might be expected, male talkers did better on the male-trained model and (with one exception) females did better on the female-trained model. (The exception shows that the different models are probably best considered vocal-tract-length dependent rather than gender-dependent. The exceptional talker was scored using the male-model results for the performance calculations of Table 5.3).

In the recognition task, the most appropriate model is *a priori* unknown so some way must be found of determining which model to use. A simple strategy is to run the recognition procedure using all models, and choose the results from the model with the best Viterbi score (the probability of the best state sequence). This has the drawback of increasing recognition cost with the number of models. Alternatively, a "pre-classifier" may be used to make an educated guess about the most appropriate model to use for recognition.

5.7.1 Classification Error vs. Recognition Performance

Recall that the decision tree may be used as a classifier by labeling each leaf with the most likely class (HMM state) in that leaf. A decision tree may be used as a *classifier* by associating a class label $\bar{C}(l_j)$ with each leaf l_j. For speech data, this may be done by

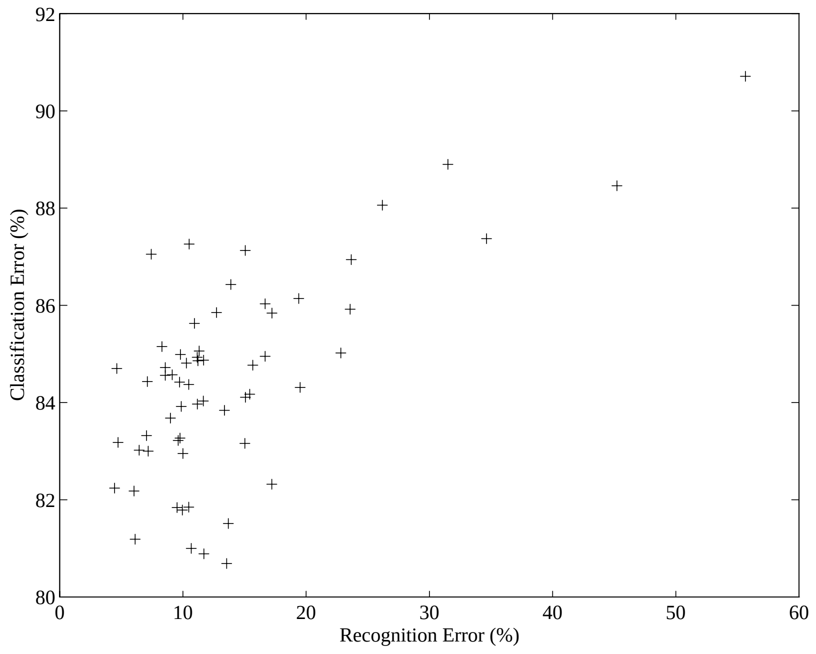

Figure 5.12: Tree classification error vs. recognition performance.

labeling each leaf with the most probable state (class) in that leaf.

$$\bar{C}(l_j) = \arg\max_i \Pr(s_i|l_j). \tag{5.3}$$

The desired probabilities $\Pr(s_i|l_j)$ may be computed from the joint probabilities in the fashion of Equation 4.9:

$$\begin{aligned} \Pr(s_i|l_j) &= \frac{\Pr(l_j, s_i)}{\Pr(l_j)} && (5.4) \\ &= \frac{\Pr(l_j, s_i)}{\sum_{i=1}^{N} \Pr(l_j, s_i)} && (5.5) \end{aligned}$$

Figure 5.12 shows the classification error versus recognition performance for 60 speech files from the talkers in the TEST data set. While the classes are not sufficiently separable to make classification particularly accurate (typically only 10 or 20 percent of vectors are correctly classified[3]), there is a significant correlation between classification error and recognition performance. As it turns out, the relative classification error is a reasonable criterion for selecting model "goodness" (the better the data fits the model, the smaller the classification error).

This is shown by the stem plots in Figure 5.11; these chart the difference between the female-model classification rate and the male-model classification rate (on an arbitrary scale). A positive result means the female model classified more vectors correctly, and thus should be a better model. The difference is strongly correlated with relative recognition performance: a negative result means superior performance on the male model and a positive result means better performance on the female model, regardless of talker gender.

5.8 Talker Adaptation

As shown in the previous section, leaf probabilities are much more important to the acoustic model than the tree structure. Because leaf probabilities are so easily trained, a promising application is talker adaptation. This is done by modifying the acoustic output probabilities to reflect a particular talker rather than the the training set ensemble, and can result in large performance gains (at the cost of talker independence). In a practical system, the amount of training information from a single talker will be small compared with the amount in a talker-independent training database. Thus talker-dependent systems are vulnerable to undertraining. The usual solution is to smooth the talker-dependent probabilities with ones from a talker-independent model. With discrete-output hidden Markov models such as the

[3]This shows why HMMs are essential: even the best classifiers are useless without some way of modeling the time variation.

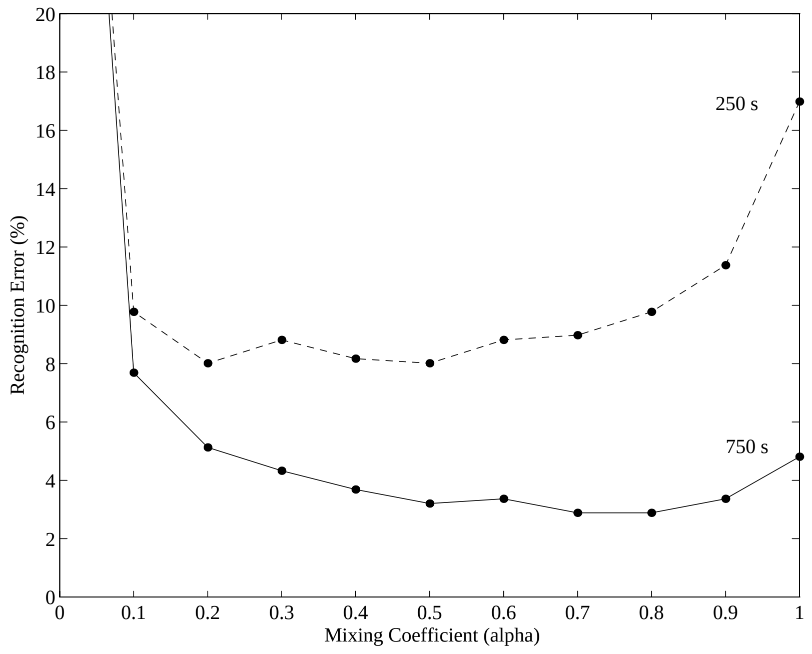

Figure 5.13: Talker-dependent Recognition Performance.

tree-based ones of Section 4.2, this may be done using a linear"mixing coefficient" α:

$$p_j^{\mathrm{adapt}}(i) = (1-\alpha)p_j^{\mathrm{indep}}(i) + \alpha p_j^{\mathrm{dep}}(i) \tag{5.6}$$

where the adapted probability is a linear mixture of the talker independent probabilities (derived from the large training database) and the talker dependent probabilities (derived solely from the particular talker).

Figure 5.13 shows the results of a preliminary talker-adaptation experiment. The same tree model used in the experiments of the previous section and Section 5.6 was trained on the FEMALE train set for the talker-independent model. (A female model was deliberately used to show the adaptation to a male talker.) The training set for the talker-dependent probabilities consisted of 140 utterances from one male talker: approximately 750 seconds, or about 12.5 minutes of additional speech. The test set consisted of 45 utterances from the same talker. The dashed line shows the effect of mixing between the talker-independent female model and the talker-dependent model trained on 250 seconds of data. A mixture coefficient of zero means that the pure female model was used while a mixture coefficient of 1 means that the purely talker-dependent model was used. Note that the mix does substantially better than either model alone. Because the talker-dependent model is undertrained (the unsmoothed $p_j^{\mathrm{dep}}(i)$ is zero for many j and i), and the talker-independent model is inappropriate (for a male talker), the combination smooths the talker-dependent probabilities by increasing the zero probabilities, resulting in better recognition performance. (This is a much better smoothing method than the "floor" of Section 5.9, because probabilities unseen in the talker-dependent training data are estimated from the talker-independent probabilities, which are almost certainly a better guess than some uniform p_{floor}.) The solid curve shows the results of mixing the talker-independent probabilities with talker-dependent probabilities trained on all 750 seconds of data. Even though the purely talker-dependent model works quite well, there is still some advantage to be gained by smoothing with the "inappropriate" talker-independent model—the error rate is reduced from 4.8% to 2.9%, a reduction of 40%. (Note that the best talker-independent error is more than four times as large.)

This method of talker adaptation should be quite practical, given the ease with which new tree models may be trained. Because one iteration of Viterbi training is sufficient to estimate the talker-dependent output probabilities, the talker-dependent model may be constructed in about the time it would take to collect the talker-dependent training data. Given the simplicity of the model mixing, an iterative method may be quite practical for "on-the-fly" adaptation. Starting with an initially talker-independent model, as each utterance from a new talker becomes available, the probabilities from the initial model are updated with the new estimates as follows:

0) Initialize probabilities $p_j^{(0)}(i)$ from talker-independent model.

1) Determine $p_j^{\mathrm{dep}}(i)$ from new utterance.

2) Update $p_j^{(n+1)}(i) = (1-\alpha)p_j^{(n)}(i) + \alpha p_j^{\mathrm{dep}}(i)$

3) Repeat 1) and 2) as new utterances become available.

5.9 Comparative Recognition Performance

As reported in Section 5.6, tree-based probability modeling can significantly enhance recognition performance when contextual information is used. While it would be instructive to compare the performance enhancement with results obtained by other researchers, it turns out that very few groups are working on the connected-alphadigit task. Much work has been done on the *isolated* alphadigits; Paliwal [65] and Boccieri & Wilpon (reported in [65]) report recognition error rates on the order of 10%. Other work at AT&T [32] has resulted in a 7% error rate on the same task using tied-mixture word-model HMMs. Cole and Fanty [25] report a 4% error rate on the isolated alphabet task, using a hierarchical recognition system that has been especially tuned to the task with knowledge-based features and neural nets.

Besides English, Jouvet et al. [50] has reported a 15.7% error rate on the connected French alphabet using continuous HMMs, while Bregler et al. [17] reports a 7.4% mean error rate on a talker-dependent connected German alphabet task using Waibel's TDNN architecture. (Neither the French or German alphabet are substantially more or less difficult than the English alphabet.)

Perhaps the best performance has been achieved by Hwang & Huang at CMU [41], who report a 12.3% word error rate on the "continuous spelling task (26 English alphabet)" with the baseline SPHYNX-II system. Though some details (such as scoring method) were not described, this result was obtained using gender-dependent "semicontinuous" (tied-mixture) HMMs where each word was represented by two models. The addition of "senonic baseforms" (essentially tied output distributions) reduced the error rate to 9.6%, which is perhaps the best published results on this task to date.

In contrast, the best recognition performance obtained here is an 11.64% error rate, using the gender-dependent tree-based models of Section 5.7 on the alphadigit task, obtained without substantial "tweaking" of any sort. Tuning codebook feature allocation and weighting, the language-model match factor, and the output probability floor would probably result in modest performance gains.

The alphadigit task may be more difficult than the alphabet because of the substantially higher perplexity. While the digits are more "orthogonal" than the alphabet, they are still a substantial source of confusion (typically, "8" is erroneously recognized as "*A*" more than 20% of the time). Though the results shown here are not directly comparable with the CMU results, they are certainly competitive; especially considering that a discrete HMM system is being compared with CMU's state-of-the-art tied-mixture recognizer.

"There's the tree in the middle," said the Rose: "what else is it good for?"

Lewis Carroll, Through the Looking Glass

Chapter 6

Future Applications and Conclusions

The importance of this work presented here may be summarized as follows. Tree-based probability models can extract contextual information from the high-dimensional space formed by feature concatenation, and can also yield useful information about the relative importance of feature-space dimensions. Tree-based models lend themselves to rapid training, which may be practical for talker adaptation. Tree-based models can achieve significantly better recognition performance than conventional minimum-distortion VQ, however, the tree-based model must be detailed and be trained with an adequate amount of training data. Previous work ([64, 2]) using decision trees for quantization has been generally unsuccessful; neither approach ultimately yielded significantly better performance than a conventional HMM system. This is perhaps because the trees used were relatively small, may have been undertrained, and did not use sufficiently wide context windows. The experiments reported in Chapter 5 indicate that both a large context window and a large tree is required to achieve better performance than a conventional VQ system.

At this point, some discussion is perhaps warranted as to why the tree-based HMM system works any better than the baseline system. Firstly, the quantization regions are determined by class distributions, not a distortion metric which is independent of the class distributions. Secondly (and perhaps most importantly), the large context windows used (30–90 ms), together with the MMI tree construction, allow the quantizer to extract useful information about feature variability, i.e., the time-varying cues that discriminate between classes (HMM states). Finally, large trees partition the feature space into many more bins than is practical with a conventional VQ system, allowing a more detailed probability model. One measure of model detail is the number of parameters: the best-performing tree models have nearly ten times more parameters than the baseline model. (It is fortunate that the LEMS speech database is sufficiently large to reliably train the tree models.)

Chapter 5 presents the results of a particular set of tree models. Given the number of independent variables in a tree model—the number of codebooks, the probability floor, the tree size and so forth—the space of possible experiments is large, and it is probable that a more optimal set of tree parameters exists. Besides the obvious "tweaks" of window size and codebook membership, there are a number of ways in which the tree-based models presented here might be improved. The "time-recurrent tree" structure of 4.4.1 is promising and deserves investigation. The partitioning of the feature space into codebooks was done in an ad-hoc manner; it would be interesting to investigate the effects of different partitions. One strategy might be to partition dimensions according to their relative importance: dimensions would be allocated to codebooks such that each codebook has relatively equal importance.

Another drawback of MMI-constructed trees is that they only take into account differences in class-conditional probabilities, and are essentially useless for modeling class-independent probabilities. This may be a drawback: consider a region in the feature space which is populated entirely by one class. Though the probability density may vary wildly in that region, further splits will not be made in the region because they do not enhance the class-discriminative powers of the tree. Thus the probability density in that region will be treated as uniform when in fact it is not. A way around this might be to introduce a "pseudo-class" that is uniformly distributed over the region. Splits that discriminate between the pseudo-class and the real class distributions will partition the node into regions of differing probabilities. If the real class distribution is uniform or nearly so, then a split will give no information and splitting may be stopped—but the cell will have an approximately uniform distribution and thus matches the uniformity assumption.

6.1 Possible Applications of Tree-based Models

Perhaps the most interesting aspect of this work is the large number of directions in which it may be pursued. Possible applications of tree-based models are discussed in this section. They include environmental adaptation, better talker independence, and investigation of different feature sets.

6.1.1 Talker Independence from "Eigentalker" Models

The performance graph of 5.11 shows that some talkers perform much worse than the average, while few perform substantially better. Tolstoy has written "all happy families are alike, but unhappy families are all different in their own way." Similarly, it mat be that "all good talkers are alike, but bad talkers are all different in their own way." Of course, bad talkers are not intrinsically bad, just talkers whose recognition results are poor. If a system was trained purely on data from that talker, there is no reason to suspect that performance would be any worse than any other talker-dependent system. This leads to an interesting speculation: suppose there are certain classes of talkers such that a system trained on one talker will do well on any other talker in the class. For the sake of argument, call a representative talker from each class an *eigentalker*. One could imagine training a number of eigentalker-dependent models, and making a talker-independent recognition system by selecting the eigentalker model (or a linear combination of eigentalker models) most appropriate to each new talker.

If the HMM output-probability distributions are viewed as clusters in feature space, the benefit of an eigentalker model is that each model should consist of relatively low-variance clusters with different means, while a talker-independent model needs a higher-variance, "smeared-out" clusters to adequately represent the different means from different talker classes.

6.1.2 Model Distance Measures

Given a tree partition of the feature space, new tree-based models may be easily trained using different data from different talkers. This results in two different probability distributions on the same feature-space partition. The "distance" between the two distributions is easily calculated by any number of metrics, for example, the mutual information (cross-entropy) of section 2.2.1. If this is done, it should provide a valuable metric as to how well a given model actually models a new talker, in much the same way that the classification error was used in section 5.7.1. Hopefully, recognition performance could be enhanced by selecting the most appropriate model on the basis of the distance measure. Possible metrics include the relative classification error, as discussed in Section 5.7.1; relative information (Kullback's *information divergence*); even the maximum forward probability computed via the Baum-Welch procedure. For unsupervised talker adaptation, the class-independent probabilities might be used.

6.1.3 Environmental Robustness

A substantial barrier to the widespread use of speech recognition systems is their lack of robustness in different acoustic environments. Merely changing the microphone can cause a state-of-the-art speech recognition system to fail miserably, even in the absence of noise or reverberation. Though no experiments were done, it may be surmised that a tree-based probability model might be more robust to environmental differences than other models, once again because tree construction is supervised: the tree is built to maximize important, information-bearing differences while other variations are presumably ignored. In addition, The talker-adaptation strategy just described could be used to enhance robustness through environmental adaptation by using different models to account for environmental variation.

6.1.4 Feature Set Ranking

Given a tree, the mutual information between the data and the classes may be calculated, which is a quantitative measure of how much useful information about the classes may be extracted from the data. The greater the mutual information, the better the feature representation. Though this has some problems (for example, undertrained models will likely have higher mutual information), it could be used as a "figure-of-merit" to investigate novel features. Feature sets having greater mutual information should result in better recognition performance.

6.2 Conclusion

The work presented here has hopefully shown that decision trees are a practical alternative to conventional minimum-distortion vector quantizers. Though not a “magic bullet,” they have interesting advantages, such as computation speed which make them suitable for real-world ASR applications. Perhaps most intriguing is the number of interesting avenues that remain to be explored.

Bibliography

[1] M. Anikst et al. The SSI large-vocabulary speaker-independent continuous-speech recognition system. In *Proc. 1991 ICASSP* [46], pages 337–340.

[2] M. Anikst, W. Meisel, M. Soares, and K. Lee. Experiments with tree-structured MMI encoders on the RM task. In *Proc. Third DARPA Speech and Natural Language Workshop* [28], pages 346–351.

[3] S. Austin, J. Makhoul, R. Schwartz, and G. Zavaliagkos. Continuous speech recognition using segmental neural nets. In *Proc. Fourth DARPA Speech and Natural Language Workshop* [29], pages 249–252.

[4] L. Bahl, P. Brown, P. de Souza, P. Gopalakrishnan, and R. Mercer. A tree-based language model for natural language speech recognition. *IEEE Trans. ASSP*, ASSP-37(7):1001–1008, August 1989.

[5] L. R. Bahl, S. Das, P. V. de Souza, M. Epstein, R. Mercer, B. Merialdo, D. Nahamoo, M. Picheny, and J. Powell. Automatic phonetic baseform determination. In *Proc. 1991 ICASSP* [46], pages 173–176.

[6] L. R. Bahl, P. V. de Souza, P. S. Gopalakrishnan, D. Nahamoo, and M. A. Picheny. Decision trees for phonological rules in continuous speech. In *Proc. 1991 ICASSP* [46], pages 185–188.

[7] L. R. Bahl, P. V. de Souza, P. S. Gopalakrishnan, and M. A. Picheny. Context dependent vector quantization for continuous speech recognition. In *Proc. 1993 ICASSP* [48], pages 632–635.

[8] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

[9] J. Bellegarda and D. Nahamoo. Tied mixture continuous parameter modeling for speech recognition. *IEEE Trans. ASSP*, ASSP-38(12):2033–2045, December 1990.

[10] R. Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, New Jersey, 1961.

[11] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the Association for Computing Machinery*, 18(9):509–517, September 1975.

[12] Michael M. Blane. Vector quantization for a talker-independent connected alphadigit speech recognizer using discrete hidden Markov models. Master's thesis, Brown University, Providence, Rhode Island, May 1990.

[13] E. L. Bocchieri and J. G. Wilpon. Discriminative analysis for feature reduction in automatic speech recognition. In *Proc. 1992 ICASSP* [47].

[14] H. Bourlard, N. Morgan, C. Wooters, and S. Renals. CDNN: a context dependent neural network for continuous speech recognition. In *Proc. 1992 ICASSP* [47], pages 49–52.

[15] H. Bourlard and C. Wellekens. Links between Markov models and multilayer perceptrons. *IEEE Trans. PAMI*, PAMI-12(12):1167–1178, December 1990.

[16] Hervé Bourlard. *Continuous Speech Recognition: from Hidden Markov Models to Artificial Neural Networks.* Ph.D. thesis, Faculté Polytechnique de Mons, Belgium, 1992.

[17] C. Bregler, H. Hild, S. Manke, and A. Waibel. Improving connected letter recognition by lipreading. In *Proc. 1993 ICASSP* [48].

[18] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, Belmont, Calif., 1984.

[19] J. S. Bridle and L. Dodd. An alphanet approach to optimising input transformations for continuous speech recognition. In *Proc. 1991 ICASSP* [46], pages 277–280.

[20] P. F. Brown. *The Acoustic Modeling Problem in Automatic Speech Recognition.* Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1987.

[21] R. Cardin, Y. Normandin, and R. De Mori. High performance connected digit recognition using codebook exponents. In *Proc. 1992 ICASSP* [47], pages 505–508.

[22] R. Casey and G. Nagy. Decision tree design using probabilistic models. *IEEE Trans. IT*, IT-30(1):93–99, January 1984.

[23] C. H. Chen. On information and distance measures, error bounds, and feature selection. *Information Sciences*, 10:159–173, 1976.

[24] P. Chou. Optimal partitioning for classification and regression trees. *IEEE Trans. PAMI*, 13(4):340–354, April 1991.

[25] R. Cole and M. Fanty. Spoken letter recognition. In *Proc. Third DARPA Speech and Natural Language Workshop* [28], pages 385–390.

[26] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The M.I.T. Press, Cambridge, Massachussets, 1990.

[27] T. Cover and J. Thomas. *Elements of information theory.* John Wiley & Sons, Inc., New York, 1991.

[28] DARPA. *Proc. Third DARPA Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania, June 1990. Morgan Kaufman Publishers, Inc.

[29] DARPA. *Proc. Fourth DARPA Speech and Natural Language Workshop*, Pacific Grove, California, February 1991. Morgan Kaufman Publishers, Inc.

[30] Lewis Carroll [Charles Dodgson]. *Through the looking glass and what Alice found there.* Macmillan, New York, 1955.

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

[32] S. Euler, B. Juang, C. Lee, and F. Soong. Statistical segmentation and word modeling techniques in isolated word recognition. In *Proc. 1990 ICASSP* [45], pages 745–748.

[33] J. Foote. The LEMS DAT/SCSI interface: User's guide and technical reference. Technical Report 95, LEMS, Division of Engineering, Brown University, Providence RI, 1991.

[34] J. T. Foote, M. M. Hochberg, P. M. Athanas, A. T. Smith, M. E. Wazlowski, and H. F. Silverman. Distributed hidden Markov model training on loosely-coupled multiprocessor networks. In *Proc. 1992 ICASSP* [47], pages 569–572.

[35] H. Gu, C. Tseng, and L. Lee. Isolated-utterance speech recognition using hidden Markov models with bounded state durations. *IEEE Transactions on Signal Processing*, SP-39(8):1743–1752, August 1991.

[36] M. Hochberg, J. Foote, and H. Silverman. A comparison of state-duration distributions for HMM-based, connected speech recognition. To appear in *IEEE Trans. Signal Processing.*

[37] M. Hochberg, J. Foote, and H. Silverman. The LEMS talker-independent connected speech alphadigit recognition system. Technical Report 82, LEMS, Division of Engineering, Brown University, Providence RI, 1991.

[38] M. Hochberg, L. Niles, J. Foote, and H. Silverman. Hidden Markov model/neural network training techniques for connected alphadigit speech recognition. In *Proc. 1991 ICASSP* [46], pages 109–112.

[39] Michael M. Hochberg. *A Comparison of State-Duration Distributions for HMM-Based, Connected Speech Recognition.* Ph.D. thesis, Brown University, Providence, Rhode Island, May 1992.

[40] X. D. Huang, H. W. Hon, and K. F. Lee. On semi-continuous hidden Markov modeling. In *Proc. 1990 ICASSP* [45], pages 689–692.

[41] M-Y. Hwang and X. Huang. Subphonetic modeling with Markov states — senone. In *Proc. 1992 ICASSP* [47].

[42] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5:15–17, May 1976.

[43] IEEE. *Proc. 1985 International Conference on Acoustics, Speech, and Signal Processing*, Tampa, Florida, March 1985.

[44] IEEE. *Proc. 1987 International Conference on Acoustics, Speech, and Signal Processing*, Dallas, Texas, April 1987.

[45] IEEE. *Proc. 1990 International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, New Mexico, April 1990.

[46] IEEE. *Proc. 1991 International Conference on Acoustics, Speech, and Signal Processing*, Toronto, Canada, May 1991.

[47] IEEE. *Proc. 1992 International Conference on Acoustics, Speech, and Signal Processing*, San Francisco, California, March 1992.

[48] IEEE. *Proc. 1993 International Conference on Acoustics, Speech, and Signal Processing*, Minneapolis, Minnesota, April 1993.

[49] F. Jelinek. Continuous speech recognition by statistical methods. *Proc. IEEE*, 64(4):532–566, April 1976.

[50] D. Jouvet, A. Lainé, J. Monné, and C. Gagnoulet. Speaker-independent spelling recognition over the telephone. In *Proc. 1993 ICASSP* [48].

[51] B. Juang and L. Rabiner. The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Trans. ASSP*, ASSP-38(9):1639–1641, Sept 1990.

[52] B. Juang, L. Rabiner, and J. Wilpon. On the use of bandpass liftering in speech recognition. *IEEE Trans. ASSP*, ASSP-35(7):947–954, July 1987.

[53] Kai-Fu Lee. *Automatic Speech Recognition: The Development of the SPHYNX System.* Kluwer Academic Publishers, Boston, 1989.

[54] Kai-Fu Lee. Personal communication, June 1993.

[55] H. Leung and V. Zue. A procedure for automatic alignment of phonetic transcriptions with with continuous speech. In *Proc. 1985 ICASSP* [43], pages 2.7.1–2.7.4.

[56] S. Levinson. Continuously variable duration hidden Markov models for automatic speech recognition. *Computer Speech and Language*, 1:29–45, 1986.

[57] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, April 1983.

[58] P. Lewis. The characteristic selection problem in recognition systems. *IRE Trans. Inform. Theory*, 8:171–178, February 1962.

[59] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Trans. Comm.*, COM-28(1):84–95, January 1980.

[60] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[61] A. Ljolje and S. Levinson. Development of an acoustic-phonetic hidden Markov model for continuous speech recognition. *IEEE Trans. ASSP*, ASSP-39(1):29–39, January 1991.

[62] A. Ljolje and M. Riley. Automatic segmentation and labeling of speech. In *Proc. 1991 ICASSP* [46], pages 473–476.

[63] H. Murveit, J. Butzberger, and M. Weintraub. Speech recognition in SRI's Resource Management and ATIS systems. In *Proc. Fourth DARPA Speech and Natural Language Workshop* [29], pages 94–100.

[64] M. Ostendorf and J. R. Rohlicek. Joint quantizer design and parameter estimation for discrete hidden Markov models. In *Proc. 1990 ICASSP* [45], pages 705–708.

[65] Kuldip Paliwal. Dimensionality reduction of the enhanced feature set for the HMM-based speech recognizer. *Digital Signal Processing*, 2:157–173, 1992.

[66] D. Paul, J. Baker, and J. Baker. On the interaction between true source, training, and testing language models. In *Proc. Third DARPA Speech and Natural Language Workshop* [28], pages 185–189.

[67] Douglas Paul. The Lincoln tied-mixture HMM continuous speech recognizer. In *Proc. Third DARPA Speech and Natural Language Workshop* [28], pages 332–336.

[68] M. Phillips, J. Glass, and V. Zue. Modeling context dependency in acoustic-phonetic and lexical representations. In *Proc. Fourth DARPA Speech and Natural Language Workshop* [29], pages 71–78.

[69] Thomas Pynchon. *Gravity's Rainbow.* Viking Penguin, New York, USA, 1973.

[70] L. Rabiner and F. Soong. Single-frame vowel recognition using vector quantization with several distance measures. *Bell Sys. Tech. J.*, 64:2319–2330, December 1985.

[71] L. Rabiner, J. Wilpon, and F. Soong. High performance connected digit recognition using hidden Markov models. *IEEE Trans. ASSP*, 37(ASSP-8):1214–1225, August 1989.

[72] L. R. Rabiner, B.-H. Juang, S. E. Levinson, and M. M. Sondhi. Recognition of isolated digits using hidden Markov models with continuous mixture densities. *AT&T Technical Journal*, 64(6):1211–1270, July-August 1985.

[73] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[74] M. Rahim. A neural tree network for phoneme classification with experiments on the TIMIT database. In *Proc. 1992 ICASSP* [47], pages 345–348.

[75] V. Ramasubramanian and K. Paliwal. Fast k-Dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Trans. SP*, SP-40(3):518–531, March 1992.

[76] S. Renals and N. Morgan. Connectionist probability estimation in HMM speech recognition. Technical Report TR-92-081, International Computer Science Institute, Berkeley, California, 1992.

[77] S. Renals, N. Morgan, and H. Boulard. Probability estimation by feed-forward networks in continuous speech recognition. Technical Report TR-91-030, International Computer Science Institute, Berkeley, California, 1991.

[78] Tony Robinson. A real-time recurrent error propagation network word recognition system. In *Proc. 1992 ICASSP* [47], pages 617–620.

[79] A Rosenfeld and A. Kak. *Digital Picture Processing.* Academic Press, Orlando, 2nd edition, 1982.

[80] R. Schwartz, Y.Chow, O. Kimball, S. Roucos, M.Krasner, and J. Makhoul. Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *Proc. 1985 ICASSP* [43], pages 1205–1208.

[81] I. Sethi and G Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Trans. PAMI*, 4(4):441–445, July 1982.

[82] H. Silverman and N. Rex Dixon. A comparison of several speech-spectra classification methods. *IEEE Trans. ASSP*, ASSP-24(4):289–295, August 1976.

[83] Harvey Silverman. On the implementation and computation of training an HMM recognizer having explicit state durations and multiple-feature-set tied-mixture output probabilities. *In preparation*, June 1993.

[84] J. Strömberg, J. Zrida, and A. Isaksson. Neural trees: Using neural nets in a tree clasifier structure. In *Proc. 1991 ICASSP* [46], pages 137–140.

[85] Lao Tzu. *Tao Te Ching.* Vintage, New York, 1972. Trans. Gia-Fu Feng and Jane English.

[86] A. Waibel, T. Hawnazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. ASSP*, ASSP-37(3):328–339, March 1989.

[87] Satosi Watanabe. Pattern recognition as a quest for minimum entropy. *Pattern Recognition*, 13(5):381–387, February 1981.

[88] C. Wellekins. Explicit time correlation in hidden Markov models for speech recognition. In *Proc. 1987 ICASSP* [44], pages 384–387.

[89] C. Wightman and M. Ostendorf. Automatic recognition of intonational features. In *Proc. 1992 ICASSP* [47], pages 221–224.

[90] J. G. Wilpon, C.-H. Lee, and L. R. Rabiner. Improvements in connected digit recognition using higher order spectral and energy features. In *Proc. 1991 ICASSP* [46], pages 349–352.

[91] Ludwig Wittgenstein. *Über Gewißheit [On Certainty].* Harper & Row, New York, 1969. Ed. G. E. M. Anscombe and G. H. von Wright.

[92] V. Zue, S. Seneff, and J. Glass. Speech database developmant at MIT: TIMIT and beyond. *Speech Communication*, 9(4):351–356, August 1990.