CLASS - A Study of methods for coarse phonetic classification

James Delmege

R. I. T.

Rochester Institute of Technology

School of
Computer Science and Technology

CLASS - A study of methods for coarse phonetic classification

By

James W. Delmege

A thesis, submitted to the Faculty of the School of Computer Science and Technology, in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science.

Approved by:

Dr. James Hillenbrand (Chairman)
John A. Biles
Peter G. Anderson

September 30, 1988

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Abstract

The objective of this thesis was to examine computer techniques for classifying speech signals into four coarse phonetic classes: vowel-like, strong fricative, weak fricative and silence. The study compared classification results from the K-means clustering algorithm using Euclidian distance measurements with classification using a multivariate maximum likelihood distance measure. In addition to the comparison of statistical methods, this study compared classification using several tree-structured decision making processes. The system was trained on ten speakers using 98 utterances with both known and unknown speakers. Results showed very little difference between the Euclidian distance and maximum likelihood; however, the introduction of the tree structure on both systems had a positive influence on their performance.

Keywords:

Artificial Intelligence
Voice Recognition
Speech Recognition
Pattern Recognition
Signal Processing
Speech Analysis

ACM:
Computing Methodologies
Natural Language Processing
Speech recognition and Understanding

Acknowledgments

I am very grateful to my committee chairman Jim Hillenbrand for his experience and expertise in guiding me through this thesis. His encouragement and advice helped keep me on track through some of the difficult times, particularly during the writing of the paper.

I would also like to thank Al Biles, my second committee member, for both his input on the research itself and his advice on getting through all of the RIT paperwork. Thanks to Rob Gayvert, who helped out with some support software and explained to me some confusing aspects of statistical analysis. My thanks also goes to Lisa Furman for her meticulous review of my drafts. For the use of their computer facilities I express my appreciation to my employer, Redcom Laboratories.

On a more personal note I would like to express my deepest thanks to my wife, Suzanne, who watched and waited many long evenings as I wrote and rewrote. Although this paper bares my name the effort was no less hers by allowing me the time needed with two small children at home.

The pursuit of education has been both a challenge and a joy to me and I would not have known either side had it not been for the encouragement of my parents, for they have shown me the importance of learning and for that I be forever thankful. Finally, I would like to express my thanks to God for the strength, love, and eternal life He has given me and my family. "The fear of the Lord is the beginning of wisdom, and knowledge of the Holy One is understanding." (Proverbs 9:10).

This research was in part supported by Rome Air Development Center, under contract number F3060285-C-0008 and done in cooperation with the Rochester Institute of Technology Research Corporation.

1 Introduction

Speech recognition has been an area of interest to computer scientists for many years. Although progress has been slow and the work often tedious, knowledge has been gained regarding what the critical attributes of speech are and how to work with these attributes in a computer. Much work has been done with the application of general pattern matching techniques producing very good success within a limited domain. The limitations placed on the speech domain for pattern matching techniques include a single speaker, a small vocabulary, and the requirement that the speaker produce words in isolation. The extension of low level pattern matching techniques to a large vocabulary, speaker independence, and continuous speech has not met with great success. For this reason, researchers have begun to look for more robust techniques that might work in more difficult domains.

Phonetic analysis is considered by some to be one of the more robust ways of analyzing speech and may prove to be successful with domains using a large vocabulary, speaker independence, or continuous speech. Phonetic analysis involves mapping the incoming speech to a sequence of phonemes. This can be done with good results by a highly trained phonetician [ZUE 79]. However, enabling a computer to perform this task is a formidable challenge. The phonetic analysis approach used here involves the basic steps of, feature extraction, segmentation, and classification. Once a set of meaningful features has been extracted, the segmentation and classification can be done by applying traditional clustering and multivariate analysis to the feature vector.

Any gross mistakes in segmentation or classification (e.g. identifying a vowel as a fricative) of the incoming voice will probably result in an incorrectly recognized word. Minor labeling errors (e.g. the fricative /sh/ labeled as /ch/) are much more likely to be tolerated or corrected at a higher level. It is this concern that has motivated the work presented in this thesis. The objective was to study ways of doing the first level segmentation and classification, placing each segment into its correct class (strong fricative, weak fricative, vowel-like, or silence). Once the correct coarse class has been established, the next level can continue the classification process down to the specific phoneme. Therefore, the primary concern of the work presented here was to ensure that the system avoid gross errors in identifying segment boundaries and phonetic labels, leaving the detailed analysis to other modules in the system.

CHAPTER 1

PHONETIC CLASSIFICATION

1.1 Phonemes and continuous speech

The human speech production system is capable of producing an almost infinite number of sounds. However, the English language is made up of approximately 42 basic sound units called phonemes. It has long been believed that each phoneme possesses certain inherent characteristics that make it distinguishable from all other phonemes. Identifying these characteristics and learning how to extract them from a speech signal is a key issue in the process of recognizing what phoneme is being spoken. Correctly performing this task is difficult due to the fact that these characteristics exhibit a high degree of variability. A major source of variability arises from interspeaker differences because each speaker pronounces phonemes in a slightly different way. Their frequency range, rate of speech, volume, accent, etc. all vary and must be taken into account. Another very difficult source of variability occurs in continuous speech and is called coarticulation. This is when the properties of a phoneme change as a function of the phonemes around it. The problem of coarticulation is best explained by viewing an example of it in a spectrogram. As shown in Figure 1.1, the words "Two" and "ten" both start with the same phoneme /t/, and although they are similar, there are also differences. The burst frequency is lower for the first /t/ than for the second, due to the anticipation of the vowel /u/. Also, notice the three occurrences of the vowel /ɛ/ (in "seven", "less", and "ten"). The second /ɛ/ is influenced by the adjacent /l/, resulting in a very low starting frequency of the second formant*. It is also influenced by the following /s/ raising the second formant upward near the end of the phoneme. The third /ɛ/ is heavily nasalized, as shown by the smearing of the first formant. These examples serve to illustrate that it can often be difficult to even see the similarities in two separate occurrences of a phoneme in continuous speech. Therefore, although there is a relatively small set of phonemes that comprise the English language, identifying them is not a simple job of matching a small set of well-defined patterns, or templates. It involves a much deeper understanding of the acoustic properties of the phonemes and their interaction with each other [ZUE 85].

* Formants are natural resonant frequencies of the vocal tract, which appear as regions of relatively high energy in spectrograms. Formant frequencies are known to carry a great deal of phonetic information.

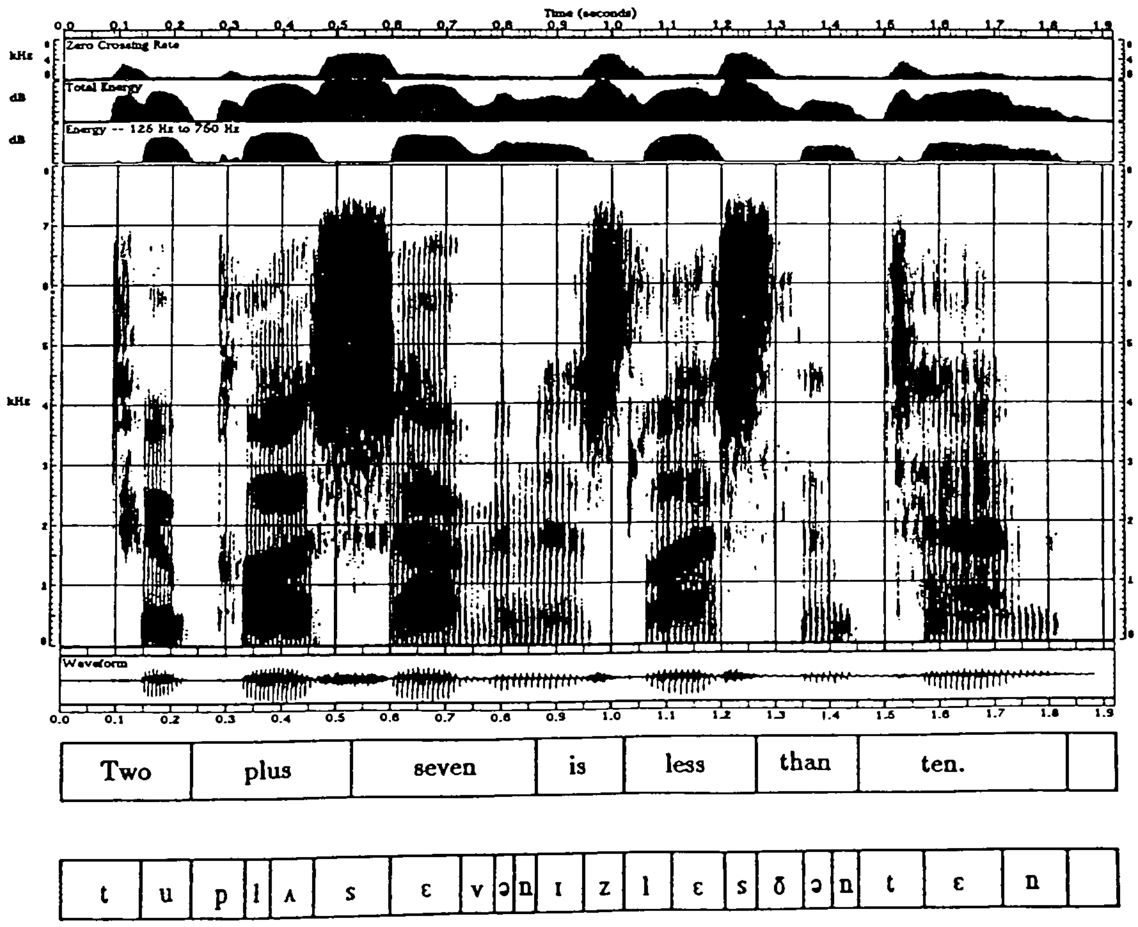


Figure 1.1 A speech spectrogram of "Two plus seven is less than ten," spoken by a male, illustrating some allophonic variations often found in continuous speech [ZUE 85].

1.2 Spectrogram-reading research

In the late 1970's, spectrogram-reading experiments were performed by Zue and Cole in which Zue was asked to read spectrograms in three categories [ZUE 79]. He was given spectrograms consisting of: (1) isolated words, (2) sensible sentences, (e.g., "The soldiers knew the battle was won."), and (3) semantically anomalous sentences (e.g., "Wake jungle gasoline sudden bright."). These utterances were unknown to Zue and were spoken by unknown speakers. His resulting phonetic transcriptions were compared with results provided by three phoenticians who listened to the utterances. Zue's segmentation* matched that of the other three phoneticians 100 percent of the time for isolated words and 97 percent for continuous speech. His segment labeling produced results from 81 to 93 percent agreement with the other phonetician's results. These results are far better than any computer speech recognition system has done thus far.

These results point have several implications for computer speech recognition. First, there is a wealth of phonetic information to be obtained directly from the speech signal. The fact that Zue performed correct segmentation 100 percent of the time with isolated words and 97 percent of the time on continuous speech says that the information needed for segmentation is present in the spectrogram. Second, the reading was based on the use of many acoustic cues, some of which were extracted and used immediately; others were not examined until a

* Segmentation is the process of determining the phonetic boundaries within an utterance.

context had been established. This indicates that the order in which the extracted features were used was significant. Finally, it was found that the sensible sentences were read almost as accurately as the semantically anomalous ones, indicating that a high level knowledge of English sentence structure is not a requirement for good phoneme recognition.

1.3 Coarse Classification

While handling the variability between speakers, one must not lose sight of the desire to recognize a large vocabulary of words. The problems associated with a large vocabulary are not just related to the sheer size of the search space, but also to the very small acoustic distinction (or distance) between words as the vocabulary size increases. For example, distinguishing reliably between "Sue" and "zoo" across a wide range of speakers can be very difficult. An approach that will help solve this problem is the use of coarse phonetic classes. Coarse classification has been used on speech projects by Leung and Zue at MIT [LEUN85]; Wilcox and Lowerre at Hewlett-Packard [WILC86]; Cole, Phillips, Brennan, and Chigier at Carnegie-Mellon [COLE86]; and Wilpon and Rabiner at Bell Laboratories [WILP85]. The coarse classes aid in breaking down a large problem into a number of sub-problems. When applied to phonetically-based speech recognition, this process generally involves categorizing speech into a few coarse classes such as, fricatives, vowels, stops, etc., then further categorizing each of those classes into their respective phonemes. The size of each coarse class may range from about three to twenty phonemes. The problems now include coarse classification and detailed classification within each coarse class.

1.3.1 Feature analysis

When attempting to identify a phoneme in a system based on phonetic features, one crucial consideration is the choice of acoustic features. The relevant features vary depending on the available information and the learning goals. For example, if the phoneme is known to be a vowel, then the formant frequencies are very important. However, formant information is not useful for differentiating among fricatives. The following sections discuss three widely used characteristics of the speech signal and how they have been used in coarse classification.

Energy

1) The presence of voice can be detected by measuring the rms energy of the signal relative to the rms energy of the noise prior to the utterance. Then, a static threshold can be set. However, if a "silent" period is known, this threshold can be set dynamically [WILC86].
2) Relative energy measures can also reveal very valuable information. Vowels and nasals can be distinguished based on the total energy relative to the peak energy in the voiced part of the signal. Also, energy in the mid passband relative to the peak energy in the mid passband during voicing can be used. The goal is to detect a loss of formant structure that is not a result of a decrease in the overall

signal energy [WILC86].

3) A comparison of energy in the low frequency range 100-350 Hz to that in the range 350-850 Hz can be useful in nasal detection [CHEN86].
4) Energy onset rate is the energy change from 1000-7000 Hz within 20 msec. To capture rapid transitions, the energy is computed every millisecond from short time Fourier transforms using a 2 msec Hamming window [CHEN86].
5) High frequency energy change, the slope of the best linear fit to the energy in the 4500-7800 Hz band over the duration of a phoneme, helps to differentiate between fricatives (which have relatively stable energy) and unvoiced plosive releases (which generally have strong onset followed by weakening aspiration) [CHEN86].

Zero crossing rate

In the detection of fricatives one critical factor is the zero crossing rate. This is the number of times the signal crosses the X-axis within a given time period (see Figure 1.2). In order not to be influenced by low background noise, there is a dead band set such that a zero crossing is only counted if the signal passes completely through this region. This dead band may be set prior to the utterance and is the amplitude of the noise seen during that time [WILC86].

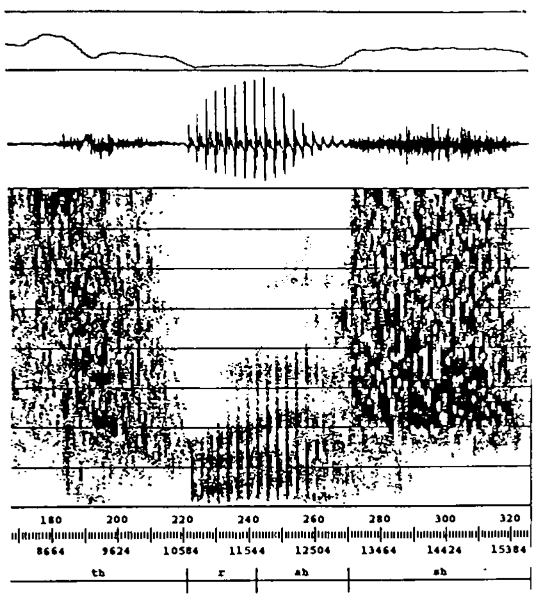


Figure 1.2 Sample zero crossing graph for the word "thrush".

Spectra

Spectral moments have been used in the analysis of voiceless obstruents [FORR88]. The moments reveal information about the distribution of energy across the frequency range. The first four moments indicate mean, variance, skewness, and kurtosis, respectively. Mean is the midpoint, or average frequency of the power distribution. Variance indicates how compressed or spread out (i.e. variable) the energy is across the frequency range. Skewness is a measure of how symmetrically the energy is distributed about the mean. Kurtosis measures the amount of energy in the ends of the spectra relative to the amount of energy in the center of the spectra range.

1.3.2 Segmentation

Once the features have been identified and extracted, the segmentation and classification steps may begin. In a recent study by Glass and Zue [GLAS87], more insight was gained into coarse segmentation and classification. In their study, the segmentation was done first, followed by classification. Segmentation involves finding the boundaries of each of the phonemes in the incoming speech signal. The algorithm used for this task was as follows:

1) The signal was divided into 10 millisecond frames.
2) A feature vector was produced for each frame.
3) A Euclidian distance measure was calculated from the current frame to each frame 10 msec to the left and to the right using the feature vectors.
4) If the distance measure changed from being closer to the left frame to being closer to the right frame, then that point was considered to be a segment boundary.

One very important aspect of this algorithm is that all of the information is local in context; that is, training data was not used, and the approach does not require preconceived ideas about segment boundaries. This is of great significance when using speaker-independent input. By changing parameters within the procedure, the sensitivity of segment detection can be altered. Very sensitive parameter settings are chosen to ensure that all true segment boundaries are found. As a result, some false boundaries are likely to be found as well. Once the initial segmentation is done, the following repetitive process is performed:

1) Each segment is associated with either its left or right neighbor using a linear distance measure applied to the feature vector of each segment.

2) Then, these larger segments are subsequently associated with one of their neighbors, and so on.

3) The merging continues until all of the segments are merged into one single segment.

The result is the segment merging diagram seen in Figure 1.3. The figure shows this process being performed on the 2.7 second utterance, "Coconut cream pie makes a nice dessert". The spectrogram and phonetic transcription are shown below the segment merging diagram. The segment merging diagram shows the steps of the repetitive process of combining segments together based on the similarity of their spectral characteristics. The very bottom of the diagram shows the results of the initial segmentation done on the 10 msec frames. Moving up the diagram, the small segments are merged together forming larger segments. The shaded areas represent the points at which this process found the correct segment boundaries.

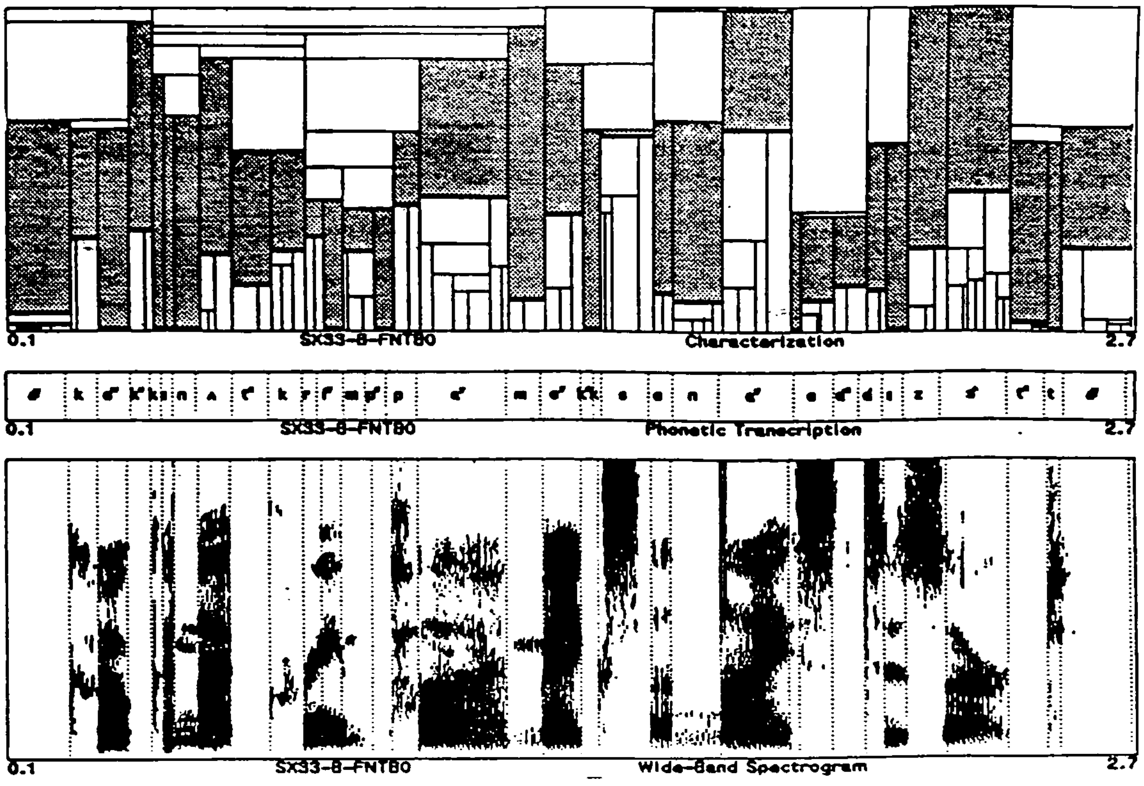


Figure 1.3 Multi-level Acoustic Segmentation, performed on the utterance "Coconut cream pie makes a nice dessert" [GLAS87].

1.3.3 Classification

Once the speech signal has been segmented, the segments can be placed into acoustic classes. The goal is to have a procedure that groups similar speech sounds into the same class and separates sounds that are different. To accomplish this goal, Glass and Zue, in their 1987 study [GLAS87], used a 500-sentence database, covering over 24 minutes of speech. Their approach is similar to that of the segmentation problem. However, instead of starting with fine segments and working toward coarse ones, they began with the assumption that all segments were in one single class, and then, iteratively, broke them into finer and finer classes. Distance measurements were taken between segments, and tighter tolerances were used to produce these finer classes. Figure 1.4 shows the steps of this process. At the bottom of the diagram, all of the segments are assumed to be in the same class. Moving up the diagram, the class breaks into two, three, four, etc. classes, until eventually, there are 61 classes at the top of the diagram. The results are not very surprising; for instance, the top two levels distinguished nearly all the consonants from vowels. The vowels were divided further based on spectral shapes corresponding to different corners of the vowel triangle. It is this coherent breakdown that is very encouraging to those considering a hierarchical structure in which to incorporate the decision process.

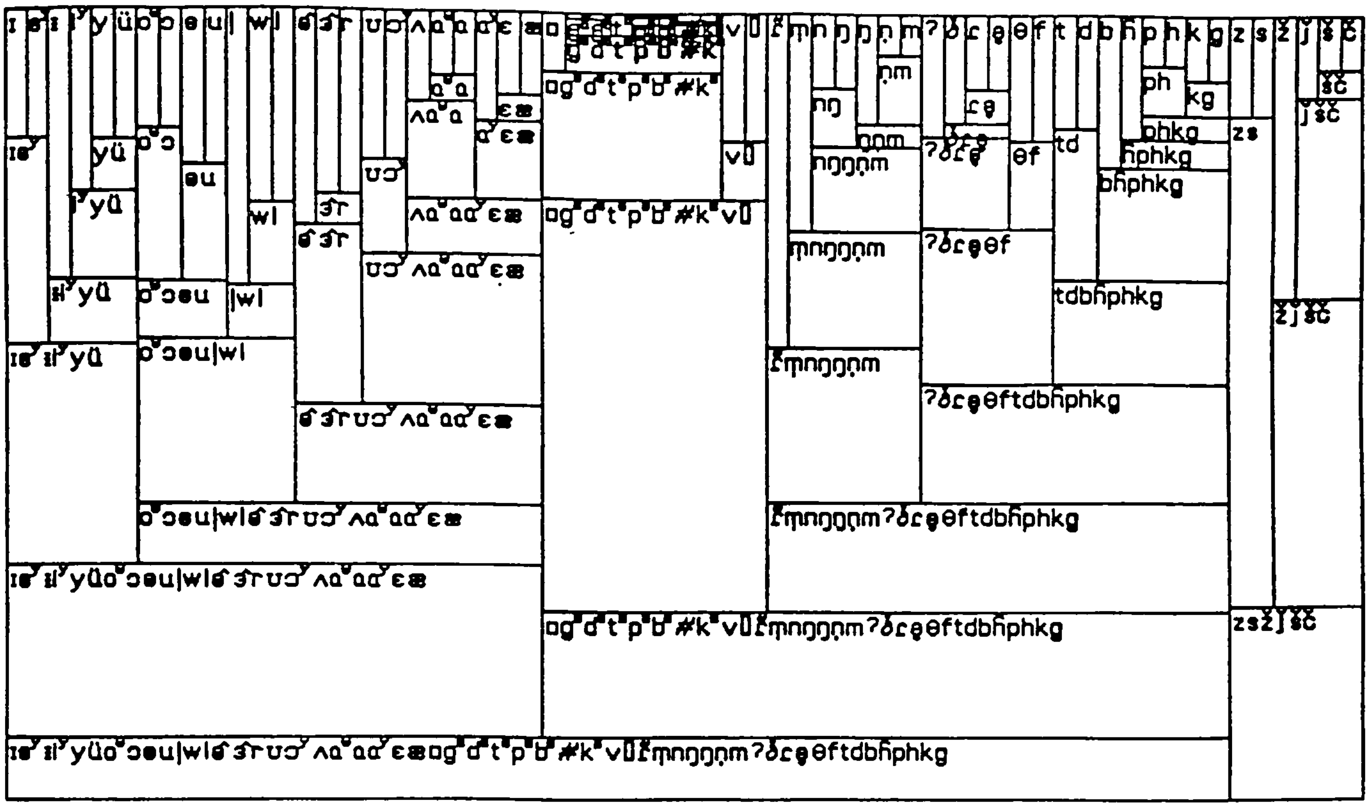


Figure 1.4 Hierarchical structure of the phonetic alphabet [GLAS87].

1.4 Structuring the decision-making process

Clearly, there needs to be a well-structured approach to the process of determining the coarse phonetic classes. A tree-structured decision making model was used to implement such a classifier at MIT by Leung [LEUN85]. The tree structure, with each non-terminal node representing a decision point and each terminal node representing a coarse class, allows different features to be examined at each decision point in order to maximize the contrast between the possible output classes. For example, Leung claims that zero-crossing rate is helpful for distinguishing sonorants from obstruents, but not for distinguishing vowels from voiced consonants. Thus, the problem of classifying the speech signal into different groups can be reduced to a sequence of sub-problems, each of which is of lesser magnitude than the task as a whole. The following sections will briefly examine several projects that have used this approach. In the following chapter, this approach will be applied to the work being done in this thesis.

1.4.1 Coarse classification for digit recognition

In a study at Hewlett-Packard Laboratories, Wilcox and Lowerre [WILC86] developed a feature-based coarse classifier. Using a very limited vocabulary (digits 0-9), it is possible to accurately hypothesize digits, with very little detail in the phonetic label. The work was done for speaker-independent applications and is applicable to large vocabularies as well. In their project, the goal was to identify the digit based exclusively on its coarse classes. The following five coarse classes were used in this process: silence, vowel, nasal-like, strong fricative and weak fricative.

Each utterance was divided into 10 msec frames, and coarse classification was performed on each frame. The next branch was determined by a Gaussian* classifier at each of the five decision points in the tree shown in Figure 1.5. At each point in the tree, different features of the signal were used as the basis for decision making.

* A Gaussian classifier is one that presumes a normal distribution of the data. The multivariate maximum likelihood distance measure discussed later is such a classifier.

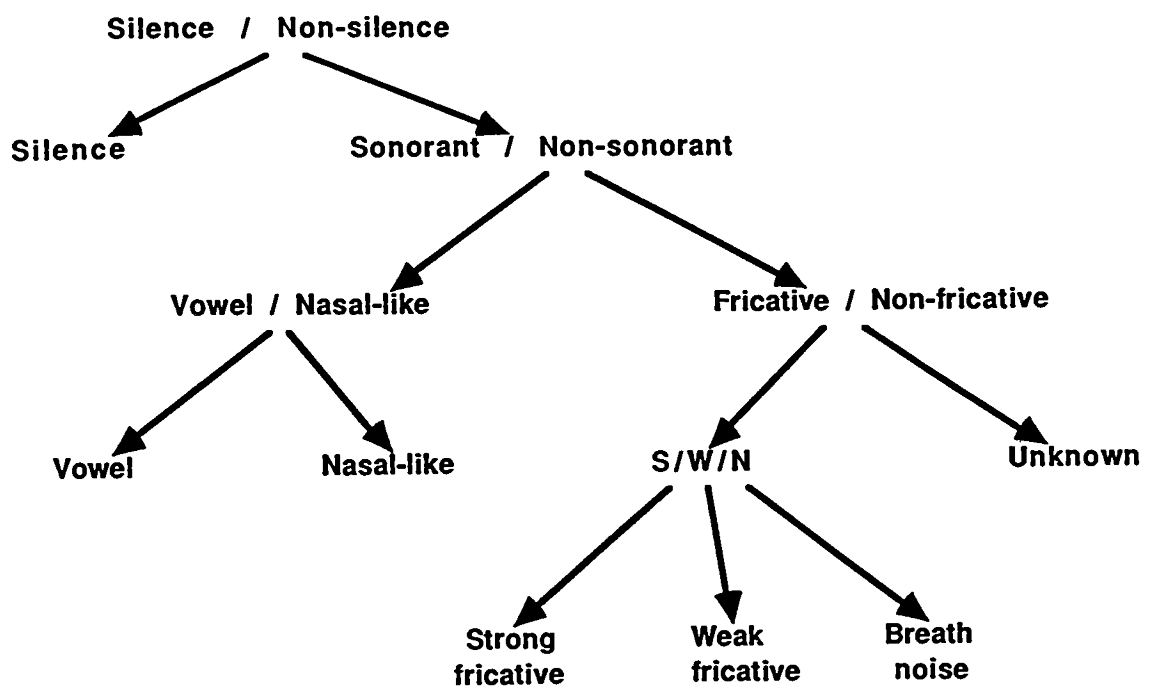


Figure 1.5 Decision tree used by Hewlett-Packard in a study on digit recognition

These results showed that without label information during training, a classifier can be trained for the sonorant/non-sonorant and the strong fricative/silence decisions. The techniques break down in distinguishing vowels from nasals and weak fricatives from strong fricatives and silence. Therefore, it was concluded that the only robust coarse classes are silence, fricative and sonorant. The sampling rate for this classifier was 12500 Hz. It was suggested, after the study, that a higher rate be used to make the strong versus weak fricative decision more accurate.

1.4.2 Coarse classification as a front end to time alignment

The tree structured approach to coarse classification was implemented in a project by Leung at MIT [LEUN85]. Leung applied coarse classification to the time alignment* of phonetic transcriptions in continuous speech using the architecture shown in Figure 1.6. Coarse phonetic classification was used to set some initial anchor points within the speech signal. From these anchor points, the detailed phonetic alignment takes shape more easily. The classifier is structured as a sequence of binary classifiers arranged in a tree. A set of classifiers was used to allow different feature sets to be used for each classifier. In this way the most salient features are used at each binary classification. The system structure is presented in Figure 1.7.

* Time alignment is the process of aligning a phonetic translation with the speech signal along the time axis.

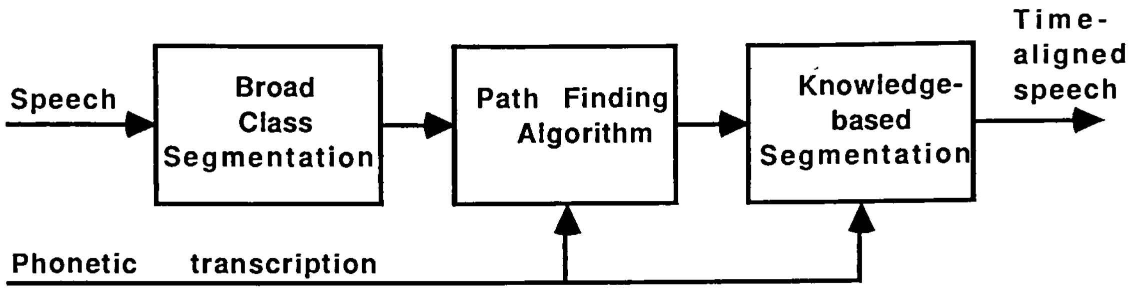


Figure 1.6 Coarse phonetic classification as used in speech time alignment by MIT [LEUN85].

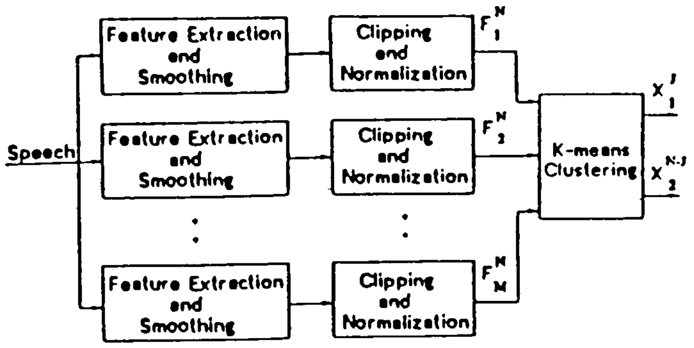


Figure 1.7 Block diagram of the use of a K-means classifier during coarse phonetic classification [LEUN85].

In the speech alignment structure, phonetic transcriptions were not used for the coarse classification. The methodology was to measure a set of feature vectors and perform a K-means clustering analysis, using a Euclidian distance measure. The results of this study indicated that this technique worked quite well for coarse classification using a small number of classes (i.e. five or six); however, it was much less successful for fine phonetic distinctions. For this reason, five classes were chosen, and each frame was assigned to one of these classes. The five classes were vowel-like sonorant, obstruent, silence, nasals and voice bars, and voiced consonants.

1.5 The K-means clustering algorithm

The main objective of a clustering algorithm is to group like tokens together and to separate different tokens. Algorithms to perform this function have existed for many years and have been applied to numerous areas of study. The K-means algorithm is a clustering technique that will form K clusters of data points in an n-dimensional space. For speech samples, the feature vector may be considered to be a point in an n-dimensional space, where each feature represents one of the dimensions. The K-means algorithm functions as follows:

Given: Many sample data points within the n-dimensional space.

Goal: To find K cluster centers (means) for the data.

Procedure:

Step 1: Choose K initial cluster centers. These may be arbitrary and are often the first K data points given.

Step 2: Distribute the remaining data points around the K cluster centers by placing each point in the cluster whose center is closest in Euclidian distance to that point.

Step 3: Calculate the center of each new cluster by choosing the point such that the sum of the squared distances from all the points within the cluster is minimized.

Step 4: If the new cluster centers are different from the old cluster centers, then repeat starting at step 2. Otherwise, the algorithm has converged, and the process may terminate.

The behavior of the K-means algorithm is influenced by the number of cluster centers specified, the choice of initial cluster centers, the order in which the samples are taken, and, of course, the geometrical properties of the data. Although no general proof of convergence exists for this algorithm, it yields acceptable results when the data exhibit characteristic pockets that are relatively far from each other. In most practical cases, the application of this algorithm requires experimenting with various values of K, as well as different starting configurations [Tou 74].

1.6 Bayes decision theory

In its simplest form, the K-means algorithm returns a mean feature vector (the cluster center) and standard deviation for each cluster. With this information, the simplest approach to categorizing an unknown data sample is to calculate a z-score for the unknown data point to each cluster and classify the data point as a member of the cluster whose z-score is the smallest.

Because speech features often result in clusters that overlap within the n-dimensional space, more sophisticated techniques are often employed to measure the distance between a data point and a cluster. If we presume that clusters of data are normally distributed around the cluster center, then we can take the mean feature vector and distribution characteristics and use a distance measure called maximum likelihood [DUDA73]. The maximum likelihood distance formula is defined as follows:

$$r^2 = (x - u)^t \, \mathrm{SUM}^{-1} \, (x - u)$$

where:

x	=	the feature vector being evaluated
u	=	the mean feature vector for the cluster (cluster center)
SUM^{-1}	=	inverse covariance matrix for the features (distribution information)
$(x-u)^t$	=	is the transpose of x-u.

CHAPTER 2
THE COARSE CLASSIFIER

2.1 Overview of the RIT Speech Understanding System

This thesis was conducted in cooperation with the Rochester Institute of Technology (RIT). At RIT there is a speaker independent, large vocabulary, continuous speech recognition project currently under development whose front end structure is shown in Figure 2.1. This system has a bottom-up architecture, meaning that the phonetic feature extractors will drive the system toward its conclusions. The objective of this part of the system is to convert the speech signal into phonetic units. The process can be broken down into the following three general steps: feature extraction, coarse classification, and fine classification.

The target machine for this project is a Sun Work-station running the Unix operating system and located at RIT. Much of the development was done off-site on an IBM PC running the MS-DOS operating system. For the CPU-intensive test runs the project was ported to a VAX 11/785 running VMS. The final project will run on all three machines, operating systems, and their respective compilers, with the source code being common to all. The code was written in C with some utility support on the

PC written in Pascal.

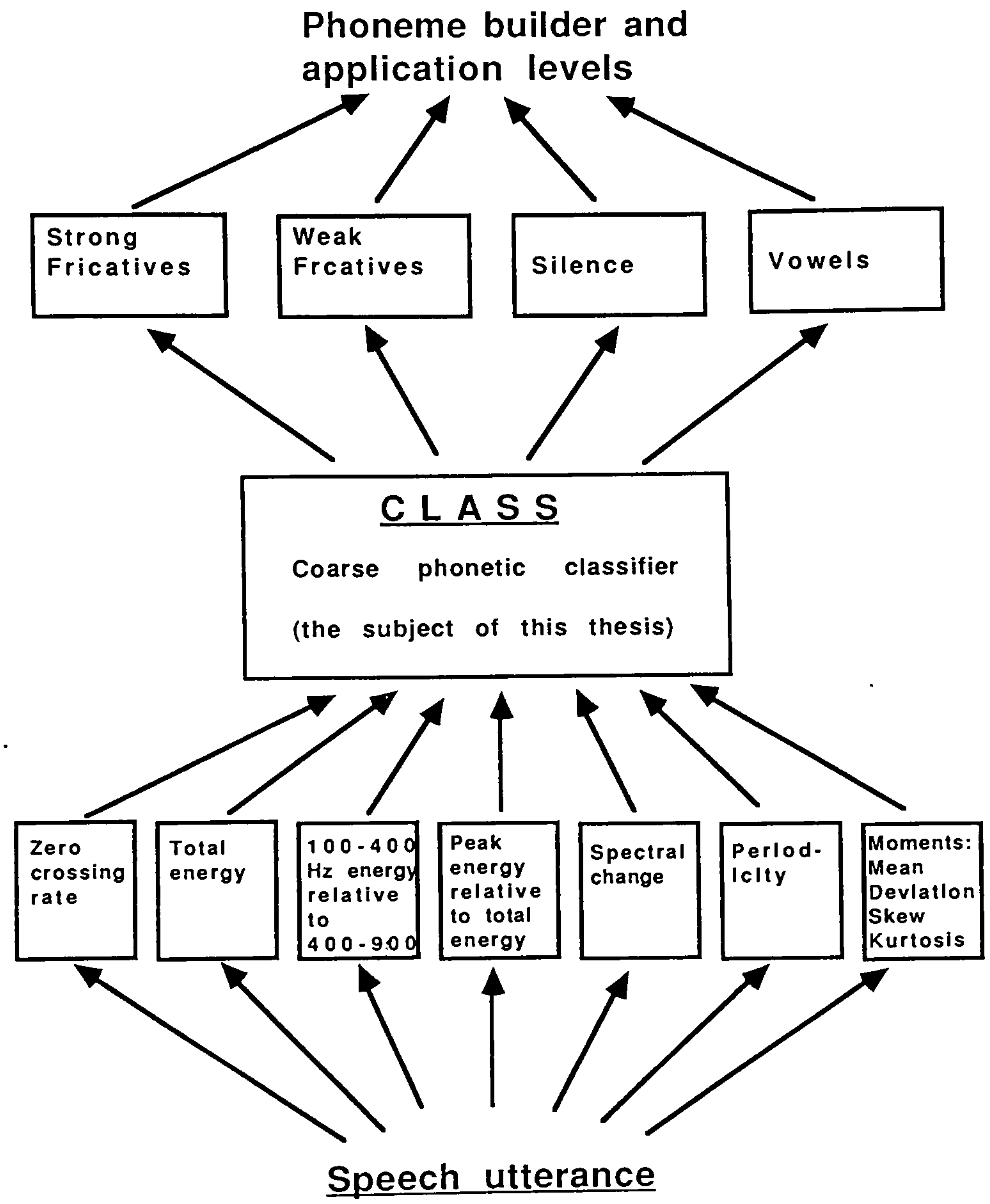

Phoneme builder and application levels
Strong Fricatives
Weak Frcatives
Silence
Vowels
CLASS
Coarse phonetic classifier
(the subject of this thesis)
Zero crossing rate
Total energy
100-400 Hz energy relative to 400-900
Peak energy relative to total energy
Spectral change
Perlod-Iclty
Moments: Mean Devlatlon Skew Kurtosis
Speech utterance

Figure 2.1 Block diagram of the structure for the speech understanding system into which this work will fit.

2.2 The objectives of this work

2.2.1 Feature Extraction

Characteristics such as zero crossing rate and energy characteristics are extracted and a feature vector is produced. This vector is the ordered set of the resulting feature values for a particular sample of the speech signal.

2.2.2 Coarse Classification

The coarse classification step involves first identifying the coarse class for each 10 msec frame of the utterance. From this, the beginning and end of each acoustic segment is identified by setting these boundaries at the point where the frames change from one coarse class to another. The segment labels will be one of the four coarse classes; strong fricative, weak fricative, vowel-like, and silence. Appendix B shows the phonetic makeup of each of these classes. The objective of coarse classification is to determine general guidelines or boundaries for the rest of the system to use.

2.2.3 Specific Classification

Once the coarse class of the segment has been hypothesized and its end points roughly marked, the signal is presented to the appropriate specific phoneme identifier. The specific phoneme identifiers are responsible for doing in depth analysis of their particular phonetic classes.

This thesis concentrated on coarse classification. The two major objectives for the work outlined in this section were: (1) to examine the merits of K-means clustering, a tree-structured architecture, maximum likelihood, and linear distance measures as they apply to coarse classification of unknown speech signals, and (2) to produce a working version of a coarse classifier. The remainder of this section will discuss the details of how these objectives were reached.

2.3 Training

Training involves systematically examining many samples of speech data in order to allow the system to learn the salient characteristics of each category. The training process used in this work will be explained in three steps.

1) Feature extraction and data preparation
2) Cluster analysis
3) The output from the training phase

2.3.1 Feature extraction and data preparation

In the first step of the training, raw samples of speech data were examined to produce a collection of label/vector pairs (LVP) (see Figure 2.2). An LVP is a two-part entity consisting of a phonetic label, which is taken from hand labeling information in the data base, and a feature vector. The feature vector is an ordered set of numbers, each number corresponding to a particular characteristic of the speech sample. The features being used are:

Zero crossing rate (ZER)
Total energy (TOT)
Relative energy (REL)
Peak energy (PEA)
Spectral derivative (SPE)
Periodicity (PER)
First moment, mean (MO1)
Second moment, deviation (MO2)
Third moment, skew (MO3)
Fourth moment, kurtosis (MO4)

```
| Label |           F e a t u r e   V e c t o r              |
+-------+----------------------------------------------------+
V       V                                                    V
{  PT,  { ZER, TOT, REL, PEA, SPE, PER, MO1, MO2, MO3, MO4 }   }
```

Figure 2.2 A label/vector pair, or LVP.

The objective of this first step of the training phase was to generate a collection of LVP's that will form the basis for the cluster analysis. Four kinds of files are involved in creating LVP's: the CMU speech data base, the options file, the executable code, and the resulting output files, which are the collection of LVP's.

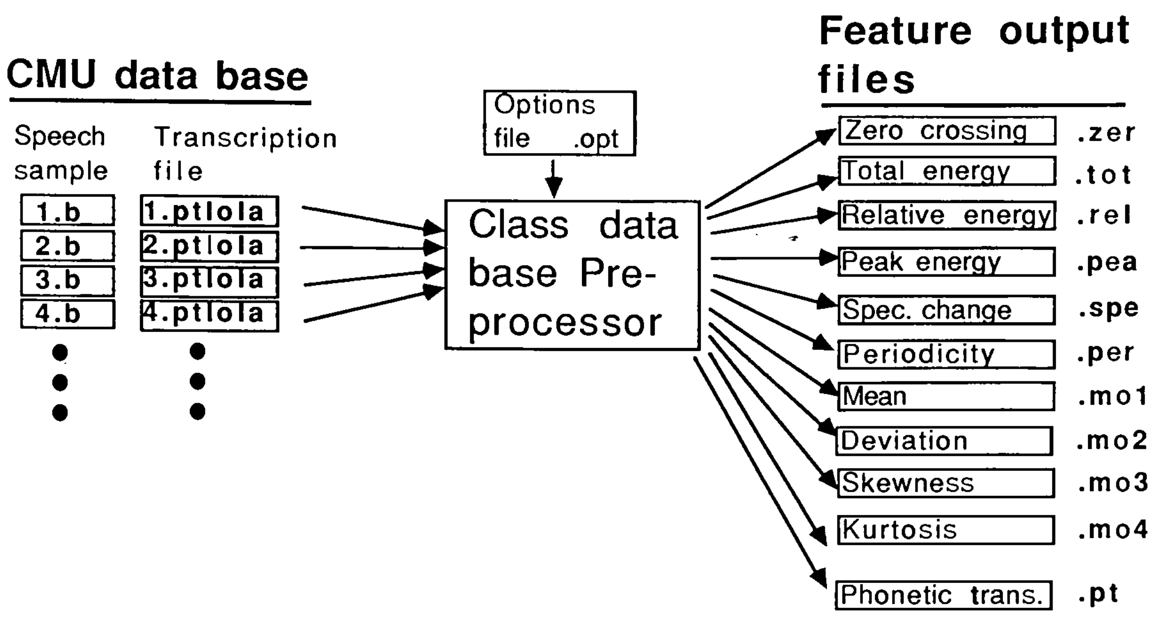


Figure 2.3 Feature extraction and data flow diagram.

2.3.1.1 The CMU data base

The speech data base used for this thesis was a subset of a data base supplied by Carnegie-Mellon University. It includes 98 utterances, spoken by 5 males and 5 females. Each utterance was roughly 3 seconds in duration, resulting in a training set of about 5 minutes. In terms of phonetic data there were about 2,300 coarse segments and some 26,600 frames. This makes the average segment 11.5 frames, or 115 msec in length. Appendix E gives the detailed list of utterances used. Each utterance has been hand-labeled with the phonetic transcription information available in a file associated with that utterance. The speech data as provided by Carnegie-Mellon University consisted of the phonetic transcription information and a binary data file of 12 bit PCM sampled at 16kHz. The data was then low pass filtered at 6kHz and downsampled to 12.8kHz. From there a 128-point FFT spectral analysis was performed once per millisecond returning 64 magnitude components of the spectra. The FFT data was then "cleaned" of background noise by removing the amount of energy found in a silent phonetic frame from all the frames. For feature extraction, all the FFT information was averaged over the 10 msec frame. This data base was then used during both the training and testing phases of this work.

2.3.1.2 The options file

The options file is a text file that contains information the preprocessor uses in order to extract and prepare the data properly. The options file was used in place of switches to the program. The format of the options file is explained in detail in Appendix A.

2.3.1.3 The preprocessor

The preprocessor section includes all of the executable code needed to extract the LVP's, as specified by the options file, and to create the output files. This code consists of a collection of programs that perform specific operations on the speech data and a final program that organizes the results and produces the output files.

2.3.1.4 The feature output files

There was a file produced for each feature being extracted and one containing the phonetic label information. The files are in binary format organized such that reading the first number from each of the files will return the first LVP, reading the second number from each file will give the second LVP, and so on. Thus, reading the eleven files into an eleven-by-N array will give N different LVP's produced from the CMU speech data base. The next phase of the training was to perform cluster analysis on these LVP's. The output files were named using the same root name as the options file along with a unique extension for each feature. The following sections discuss the implementation of each of the features.

Phonetic transcription file.PT - This file contains the phonetic transcription information. It indicates what phonetic label was given to the speech sample by human interpretation. This will be considered the most correct phonetic label and will, therefore, be used in determining how well the clustering techniques perform.

Zero crossing rate file.ZER - This file contains zero crossing rate information. This is the number of times the speech signal crosses completely through the dead band around the zero line for the duration of the frame. The limits of the dead band may be changed in the options file.

Total energy file.TOT - This file contains the total energy in each frame.

Relative energy file.REL - This file contains a comparison of the energy in the 100-400 Hz range to the energy in the 400-900 Hz range. The frequency ranges can be altered in the options file.

Total to peak energy file.PEA - This file contains the total energy relative to the peak energy. The peak energy is determined by finding the FFT component containing the most energy. Then, the ratio of that component to the total is calculated.

Spectral change file.SPE - This file contains the spectral change. This is the rate of change of the energy from one frame to the next. It is calculated by summing the differences of all the components of the FFT between two adjacent frames.

Periodicity file.PER - This file contains the periodicity calculation. Periodicity is a measure of how periodic the waveform is, which helps separate signals such as fricatives from vowels.

Mean file.MO1 - This file contains the first moment, mean. The mean is calculated by:

```
MO1  = (1) * rms[1] +   . . .  + (64) * rms[64]
```

where rms[1] is the magnitude of the rms energy for the signal from 0 to 100 Hz., rms[2] is the energy from 100 to 200 Hz, etc.

Deviation file.MO2 - This file contains the second moment, variance. The variance is calculated by:

```
MO2  = (1-MO1)^2 * rms[1] +   . . .  + (64-MO1)^2 * rms[64]
```

Skewness file.MO3 - This file contains the third moment, skewness. The skewness is calculated by:

```
MO3  = (1-MO1)^3 * rms[1] +   . . .  + (64-MO1)^3 * rms[64]
MO3  = MO3 / MO2^3/2
```

Kurtosis file.MO4 - This file contains the fourth moment, kurtosis. The kurtosis is calculated by:

```
MO4  = (1-MO1)^4 * rms[1] +   . . .  + (64-MO1)^3 * rms[64]
MO4  = ( MO4 / MO2^2 ) - 3
```

2.3.2 Cluster analysis

Once the feature information has been extracted and the LVP's collected, the clustering began. This involved assigning each LVP to one of the coarse classes. The collection of all the LVP's assigned to any one class is called a cluster. The goal of this phase was to split the collection of LVP's into clusters such that each cluster represents one of the coarse classes being analyzed.

2.3.2.1 The tree structure

The determination of which class an LVP should be assigned to could be done in many ways. No matter what technique is used, there must be an underlying structure to the decision making process. This structure can be as simple as evaluating all features together and making a single decision, or it can be broken down into a series of smaller decisions, evaluating a subset of the features during each decision. Either structure can be represented by a tree where each terminal node represents a coarse class, and the branches off of a non-terminal node represent the decision that must be made. The three trees shown below represent the extremes of the tree structures for a decision tree with four coarse classes and was the basis for comparison in this work. The main objective was, to determine the effects (positive or negative) on the classifier of making several consecutive decisions as opposed to one comprehensive decision.

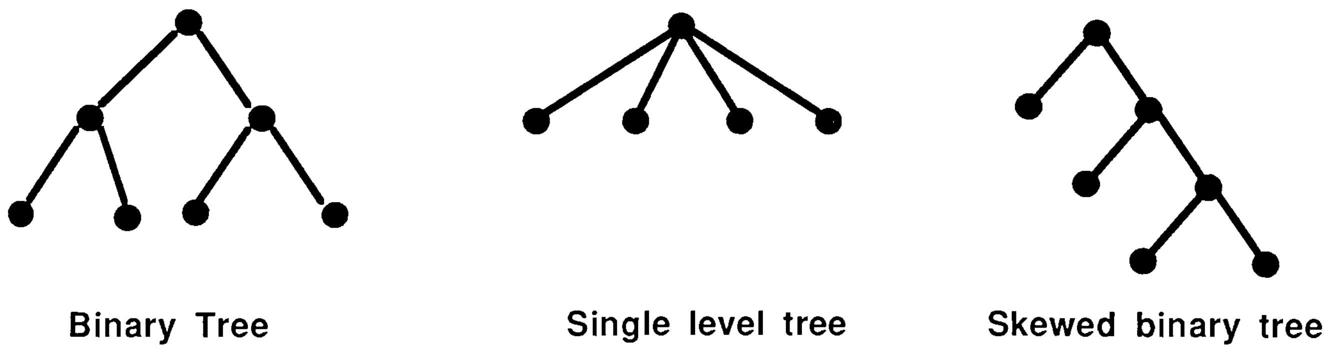


Figure 2.4 The three tree structures being evaluated in this study.

With the tree-structured decision process, the goals of the clustering phase were to determine the clusters of LVP's at each node in the tree and to determine which features should be used at each of the non-terminal nodes, in the tree. There were two techniques used to determine the answers to these questions.

2.3.2.1.1 Clustering the data

The simplest and most obvious clustering technique is to simply place the LVP's into one of the terminal nodes, based on its phonetic label. With the terminal node established, all the non-terminal nodes that must be passed through are also established and thus the clusters at each node are known. The advantage of this approach is that it allows the clusters to most completely represent each class forming logical phonetic classes. The drawback is that you must determine in advance the mapping of phonemes to coarse classes and this is not a simple task.

A second approach uses only the feature vector to separate the data into clusters. The K-means algorithm, described earlier, was applied to perform the clustering. All the LVP's were initially placed in the root node cluster. The K-means clustering algorithm then separates the LVP's between the children of the root node. The process was repeated on each of the children until all of the feature vectors were assigned to one of the terminal nodes. At each of the non-terminal nodes, the K-means clustering algorithm was run once for each combination of features. A performance index, the percent of feature vectors correctly classified as specified by the phonetic label, was determined. A difference between this and the previous clustering method is that previously the clusters were based solely on the phonetic label. The K-means technique uses the phonetic label to determine the performance of the features, but it does not use it to determine the final assignment of the LVP to a cluster. Cluster assignment is based solely on the feature vector. Figure 2.5 shows the collection of LVP's as they progress through the tree structure during this method of training.

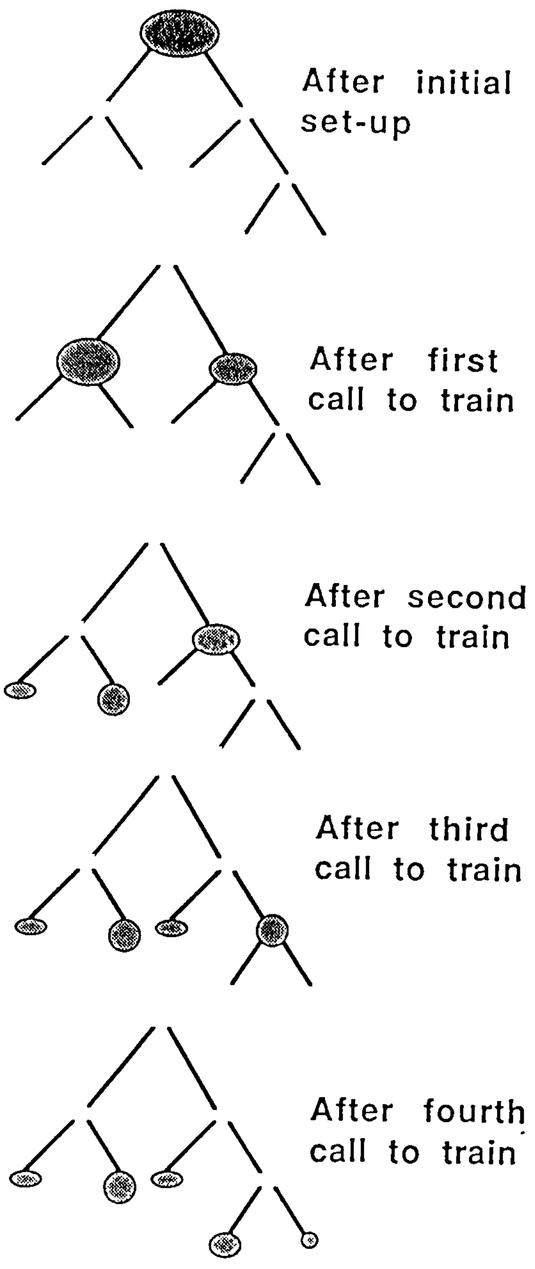

After initial set-up
After first call to train
After second call to train
After third call to train
After fourth call to train

Figure 2.5 Data clusters as they flow through a training session using K-means.

2.3.3 The output from the training phase

In summary, the training phase trained the system on three different tree structures, each using two different techniques. This yields the following six combinations of training results that will be compared to each other.

1)	Maximum likelihood	- Binary tree
2)	Maximum likelihood	- Single level tree
3)	Maximum likelihood	- Skewed binary tree
4)	K-means	- Binary tree
5)	K-means	- Single level tree
6)	K-means	- Skewed binary tree

For each combination of techniques, the training session produced statistics associated with the cluster at each node, with the exception of the root node. For the maximum likelihood training sessions, the statistics included a mean feature vector and an inverse covariance matrix. For the K-means training sessions, the characteristics were the mean feature vector and standard deviation.

2.4 The classification of an unknown data sample

Once the training was completed the classification stage is rather simple. To classify an unknown sample of data, the system performed the following steps:

1) Start at the root node.
2) Compute the feature vector of the unknown speech frame.
3) Calculate the distance between the computed feature vector and the cluster centers associated with each of the children.

4) Move down in the tree to the child whose cluster center is closest* to the unknown frame.

5) If the new node is a terminal node, then this determines the coarse class of the phoneme. If not, then return to step 3.

The output of the classification phase includes the coarse class, or terminal node, in which the frame has been classified. It was also advantageous to know the second most likely class in which the frame falls, and the probability with which the frame falls into each class. Under the techniques using maximum likelihood, the distance measure can be converted directly to the probability that the unknown frame is a member of the cluster. Therefore, this result can be returned directly as the probability measure. With the K-means technique, a probability is not generated directly; and if one is to be returned, it must be calculated, the maximum likelihood technique returned a probability. The first and second choices will be determined by taking the first and second best distance measures at the last non-terminal node visited.

* When trained with the K-means technique, a z-score is used to measure closeness. When trained with maximum likelihood, the distance used is the one returned directly from the formula.

CHAPTER 3

RESULTS

3.1 Training results

At the initial design of this project it was intended that the systems would be able to distinguish between the five classes: vowel-like, strong fricative, weak fricative, silence, and voiced closures. Preliminary testing suggested that the distinction between silence and closures could not be made reliably. Figure 3.1 shows sample spectrograms of silence (bg) and three different voiced closures (bcl, dcl, thcl). From these samples, the similarity between the silence and closure classes is evident. Therefore, the classification results reported here are scaled down to include only the final four classes: silence, strong fricative, weak fricative, and vowel-like. The class vowel-like could be described as "everything else", or voiced sounds. Figure 3.2 shows a sample spectrogram of each of the four classes.

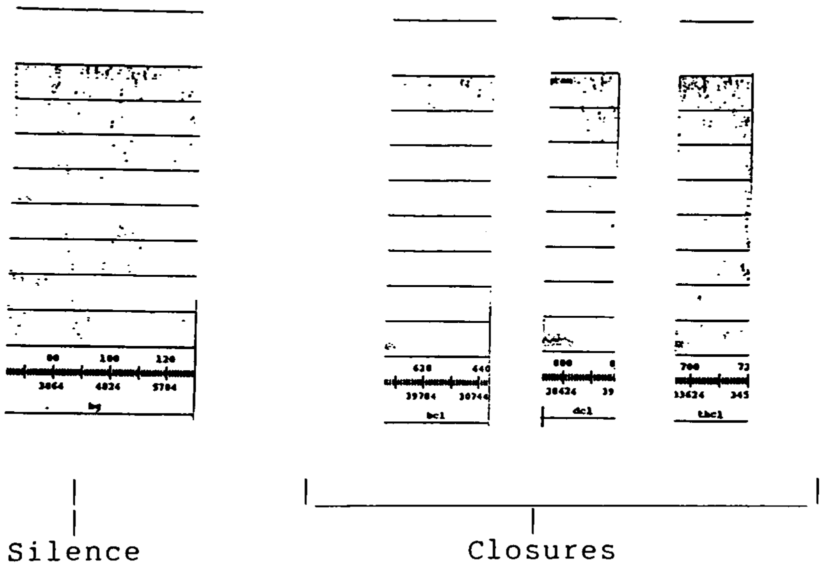


Figure 3.1 Sample spectrograms of the small difference between the silence and closure phonemes.

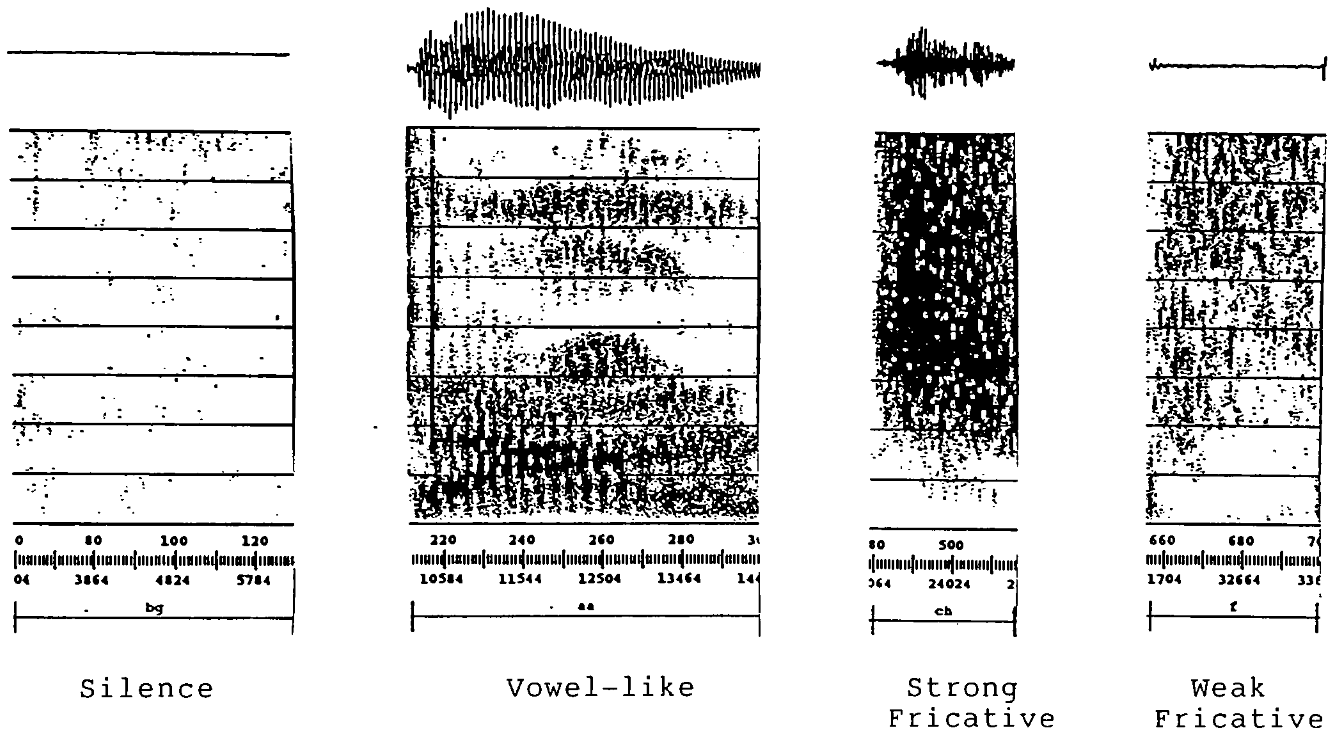


Figure 3.2 Examples spectrograms of the four coarse classes being identified in this study.

The results will be examined in the same order in which they were carried out, starting with clustering and ending with overall performance comparison. Throughout the presentation of the results the following abbreviations will be used:

Sil	- The class "silence"
SF	- The class "strong fricatives"
WF	- The class "weak fricatives"
Vow	- The class "vowel-like"

K1	- K-means training, single level tree
KB	- K-means training, full binary tree
KS	- K-means training, skewed binary tree
M1	- Maximum likelihood training, single level tree
MB	- Maximum likelihood training, full binary tree
MS	- Maximum likelihood training, skewed binary tree

zer	- Zero crossing rate
tot	- Total energy
rel	- Relative energy
pea	- Peak energy
spe	- Spectral change
per	- Periodicity
mo1	- 1st moment, mean
mo2	- 2nd moment, deviation
mo3	- 3rd moment, skewness
mo4	- 4th moment, kurtosis

3.1.1 Cluster analysis

The cluster analysis was performed by using the K-means method at all decision points in each of the three tree structures. The results of the clustering are shown in Table 3.1. These results show that the clusters were very similar regardless of the tree structure being used. Because K-means is inherently a clustering algorithm, this step was only done using K-means. The classes produced by K-means were also used as the class sets for maximum likelihood.

Table 3.1 Coarse phonetic clusters as produced by K-means on each of the tree structure used in this study.

	Single level tree	Binary tree	Skewed tree
Silence:			
	sil pau pcl	sil pau pcl	sil pau pcl
	tcl kcl qcl	tcl kcl qcl	tcl kcl qcl
	gcl dhcl bcl	gcl dhcl bcl	gcl dhcl dcl
	q	dcl hh q	q dh v
		h p g	
		k	
Strong fricatives:			
	ch sh jh	ch sh jh	ch sh jh
	zh s z	zh s z	zh s z
	t t-h	t t-h	t t-h
Weak fricatives:			
	hh h th	f th k-h	p g h
	f p g		k f th
	k		hh p q
Vowel-like:			
	uh ao aa ey ay	uh ao aa ey ay	uh ao aa ey ay
	aw ow e o ih	oy aw ow e o	oy aw ow e o
	eh ae ah dh dcl	ih eh ae ah m	ih eh ae ah m
	m n ng em eng	n ng em w eng	n ng em w eng
	dx ix uh ux oe	uw dx ix uh ux	uw dx ix uh ux
	ax ah r er axr	oe ax ah r er	oe ax ah r er
	el l iy y w	y el l iy axr	y el l iy axr
	uw v	dh v	

3.1.2 Tree structures

With only four classes, there is a very small set of tree structures that can be used for the classification. However, there is the problem of determining which classes should be at each terminal node in the tree. Figure 3.3 shows the three structures used and the classes that were chosen for each terminal node in the trees. The non-terminal nodes are labeled with a "D" followed by a number. This is the decision point number, and these nodes will be referred to by that number. When choosing the class positions for the binary tree the choice of strong and weak fricatives together seems like a very reasonable one. Tests that were run on the other combinations, indicated that this arrangement was optimal. In the skewed tree structure, the goal is to find the most easily identifiable class first, then the next easiest, and finally to split the last two as best as possible. For the skewed tree, tests were also run with each of the classes in each different position, and the best performer was the combination shown in Figure 3.3.

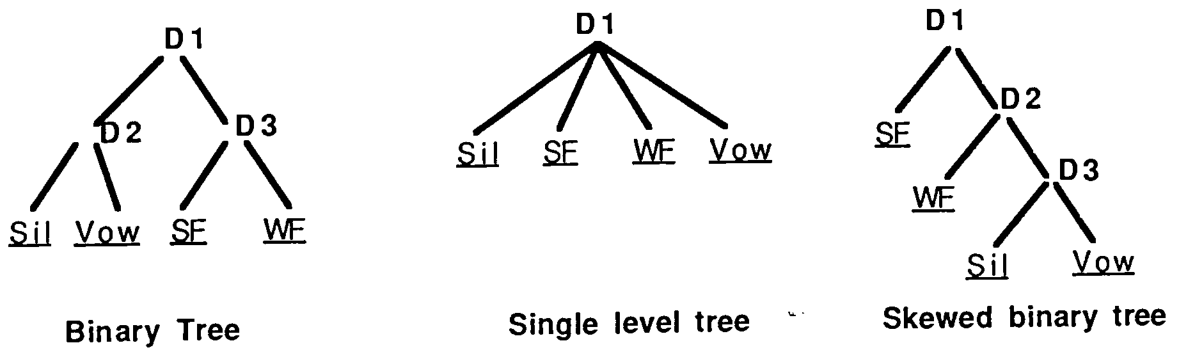


Figure 3.3 Tree structures shown with their final classes.

3.1.3 Feature Utilization

Having set the tree structures and classes, the next step was to determine the best set of features to be used at each decision point in each tree. This was done by making a test run of all combinations of the features at each decision point and choosing the feature set that made the least number of classification errors. This was done for both the K-means and maximum likelihood methods on all three tree structures, which resulted in performing classification analysis 14,336 times on the data set of around 26,000 frames. The results of these tests are shown in Table 3.2.

Of the total of ten features, the ones most often used by the top performing classifiers were zero crossings(zer), total energy(tot), and periodicity(per) Followed by relative energy(rel), mean(mol), deviation(mo2) and kurtosis(mo4). The least used of all the features were peak energy(pea), spectral change(spe) and skewness(mo3).

Table 3.2 Best features to use at each decision point in the trees shown in Figure 3.3. Results were determined after trying all combinations of all features at each decision point.

decision point:	D1	D2	D3
K-means single level tree	zer tot pea rel per mo1 mo3 mo4		
K-means binary tree	zer tot rel spe per mo1 mo2 mo3 mo4	zer tot pea per mo4	tot spe mo1 mo2 mo3 mo4
K-means skewed	zer rel per mo2	zer tot per	zer tot pea rel per mo4
Max. single level	tot rel per mo4		
Max. binary tree	tot pea per mo2	zer tot per mo1	zer tot rel spe per mo1 mo2 mo3 mo4
Max. skewed	per mo2	zer tot pea mo1	zer tot per mo2

3.2 System comparison and evaluation

One of the goals of this work was to identify the best system out of the six different ones constructed. The measures that will be used to represent performance comparisons are the following:

1 Percentage of correctly classified frames out of all frames.

2 Percentage of correctly classified frames disregarding frames that are within 10 msecs of a segment boundary.

3 Percentage of correctly classified segments to within 10 msecs of the true segment boundary.

4 Percentage of correctly classified segments anywhere within the true segment.

The first two measures represent performance analyzed on a per frame basis, (i.e. each 10 msec frame is evaluated independently of its neighbors). The first measure is an overall indication of the performance of the system. Here, each 10 msec frame is marked either right or wrong based on the hand-labeled information. This measure represents the number of frames correctly classified relative to the total number of frames in the test set. The second measure is also a ratio of correct to total frames; however, in this figure, frames that are within 10 msecs of a segment edge are ignored. It has been shown that hand-labeling is accurate to only about 10 msecs [SMIT88]. An examination of these two measures indicates where the errors are being made. For example if the first is lower, then the errors are near the edges, if it is higher, then they are in the centers, and, if they are the same, then the errors are uniformly distributed throughout the segments.

The third and fourth measures represent performance with respect to segments rather than frames. A segment is a sequence of frames all having the same class. For example, an 80 msec noise associated with the phoneme /s/ would create a strong fricative segment of 8 frames. Performance measure three indicates the percent of correct segments, where a correct segment is defined as an instance in which the classifier correctly labeled any of the frames within the segment. Measure four tightens up this performance figure by defining a correct segment, as an instance in which all but the boundary frames are correctly classified. Figures 3.4 and 3.5 show several examples of these results.

```
      Example frame-by-frame type 1 analysis
===============================================================
  True classes:  1 1 1 1 3 3 3 3 3 3 1 1 1 1 1 2 2 2 2
   Classifiers:  0 0 1 0 0 3 3 3 3 3 3 0 0 2 2 2 2 2
   Frame match:  n n y n n y y y y y n n n n n y y y y = 10 yes's
                                                          out of 19
                                                          = 53%


      Example frame-by-frame type 2 analysis
===============================================================
  True classes:  1 1 1 1 3 3 3 3 3 3 1 1 1 1 1 2 2 2 2
   Classifiers:  0 0 1 0 0 3 3 3 3 1 3 3 0 0 2 2 2 2 2
   Frame match:  * n y * * y y y y * * n n n * * y y * = 7 yes's
                                                          out of 11
                                                          = 64%


  Notation:  y = "yes" the frames (or segments) match.
             n = "no" the frames (or segments) do not match.
             * = "do not care", this frame is an edge and
                 not considered in the analysis.
```

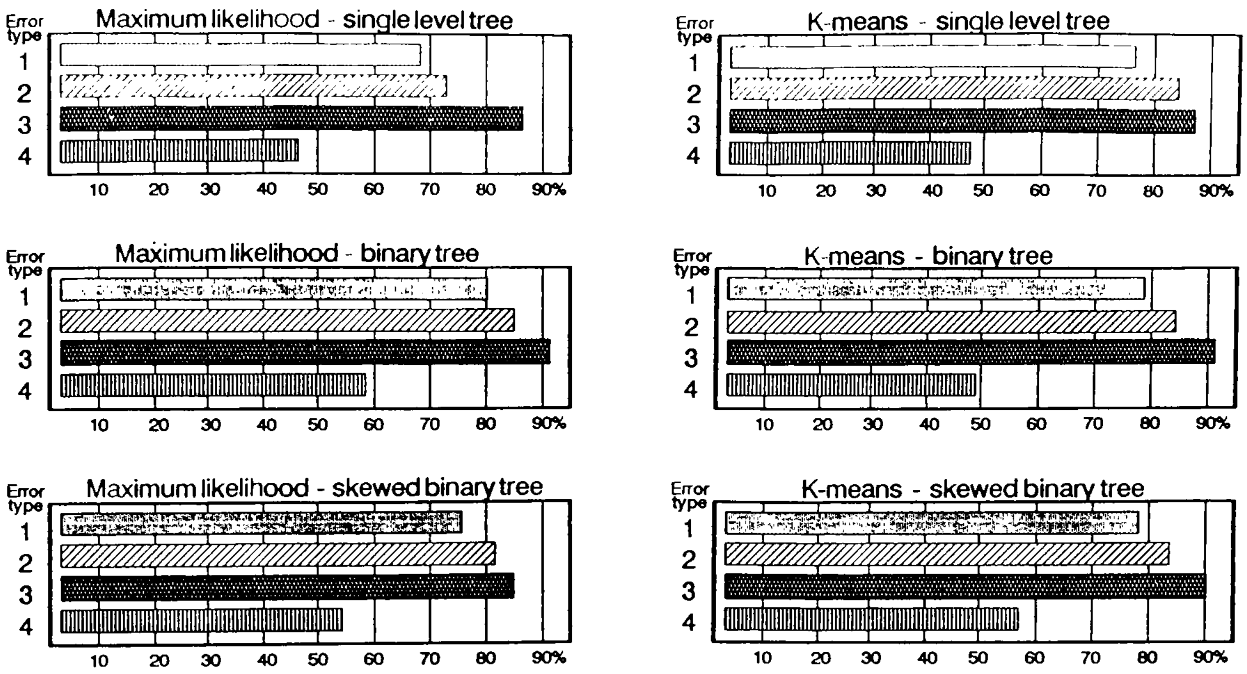
Figure 3.4 Examples of frame-by-frame analysis

```
        Example of segment-by-segment type 3 analysis
=============================================================
 True classes: |1 1 1 1|3 3 3 3 3 3|1 1 1 1 1|2 2 2 2|
  Classifiers: |0 0 1 0|0 3 3 3 3 3|3 3 0 0 2|2 2 2 2|
  Frame match: |n n y n|n y y y y y|n n n n n|y y y y|
Segment match: |   y   |     y     |    n    |   y   | = 3 yes's
                                                       out of 4
               (segment marked "y" if any frame        = 75%
                within the segment is correct.)


        Example of segment-by-segment type 4 analysis
=============================================================
 True classes: |1 1 1 1|3 3 3 3 3 3|1 1 1 1 1|2 2 2 2|
  Classifiers: |0 0 1 0|0 3 3 3 3 1|3 3 0 0 2|2 2 2 2|
  Frame match: |* n y *|* y y y y *|* n n n *|* y y *|
Segment match: |   n   |     y     |    n    |   y   | = 2 yes's
                                                       out of 4
               (segment marked "y" if all              = 50%
                but edge frames are correct)
```

Figure 3.5 Examples of segment-by-segment analysis

Figure 3.6 shows the performance results of the six classifiers after classifying all 96 utterances, and Appendix D shows the full set of confusion matrices and tables from which this summary is constructed.



Error types:

1 Frames labeled correct out of all frames.
2 Frames labeled correct disregarding boundary frames.
3 Segments correct to within 10 msecs of boundary.
4 Segments correct anywhere within the true segment.

Overall Performance Ratings

Rank	Method		Tree Type
1	Maximum likelihood	-	Binary tree
2	K-means	-	Binary tree
3	K-means	-	Skewed binary tree
4	K-means	-	Single level tree
5	Maximum likelihood	-	Skewed binary tree
6	Maximum likelihood	-	Single level tree

Figure 3.6 Performance for each of the six classifiers in four different areas, and there relative ranking.

From these classification performance figures, the best overall classifier was the maximum likelihood binary tree system. It was the top performer, or tied for the top, in all four areas. The last column shows the overall ranking of each classifier from best to worst. These overall are based on the sum of the rankings for each of the four performance measures. The lowest number is considered to be the best classifier and is assigned to one; the next lowest is given a two, etc. Of the tree structures, the binary tree performed best since two systems using it outperformed all of the other systems. Of the two distance methods, the K-means, or simple Euclidian distance, performed better overall, holding three of the top four positions. However, maximum likelihood in combination with the binary tree structure was the best classifier, indicating an interaction between the tree structure and the distance measure.

3.3 Speaker independence

The system was tested on five male speakers and five female speakers with nine or ten utterances from each speaker. To measure sensitivity to different speakers, the system was trained and tested on all combinations of the groups of male and female speakers. For example, it was trained on all speakers and then tested on just males, then it was trained on females and tested on just females, etc. Table 3.3 shows the results for K-means using a binary tree, which produced results typical of all the systems tested. The numbers in the table are performance measures 1 and 2 from the list in section 3.2, (percentage correct of all frames and percentage correct of frames disregarding frames within 10 msecs from a segment edge).

Table 3.3 Classification results of training and testing all the combinations of male, female and both. These performance results were from K-means using a binary tree, and the figures shown are error types 1 and 2 as described in Figure 3.6.

Training set ↓ / Test set →	All	Male	Female
All	79% (84%)	77% (84%)	80% (86%)
Male	78% (84%)	77% (84%)	79% (84%)
Female	79% (85%)	77% (83%)	81% (87%)

3.4 Visual Inspection of the Results

In concluding the presentation of results, it is valuable to actually see the manner in which the classifiers function for particular utterances. Figures 3.7 and 3.8 show the results of the six classifiers on two sample utterances with the corresponding spectrograms. These graphs are a representative sample of the type of results produced by the six classifiers. From these figures, it can be seen that most of the classification errors are taking place at the class boundaries. This reaffirms what was seen in the performance indicators shown in the last section. The typical errors are those such as extending or shortening a segment and are of the type that might be dealt with at higher levels of smoothing software.

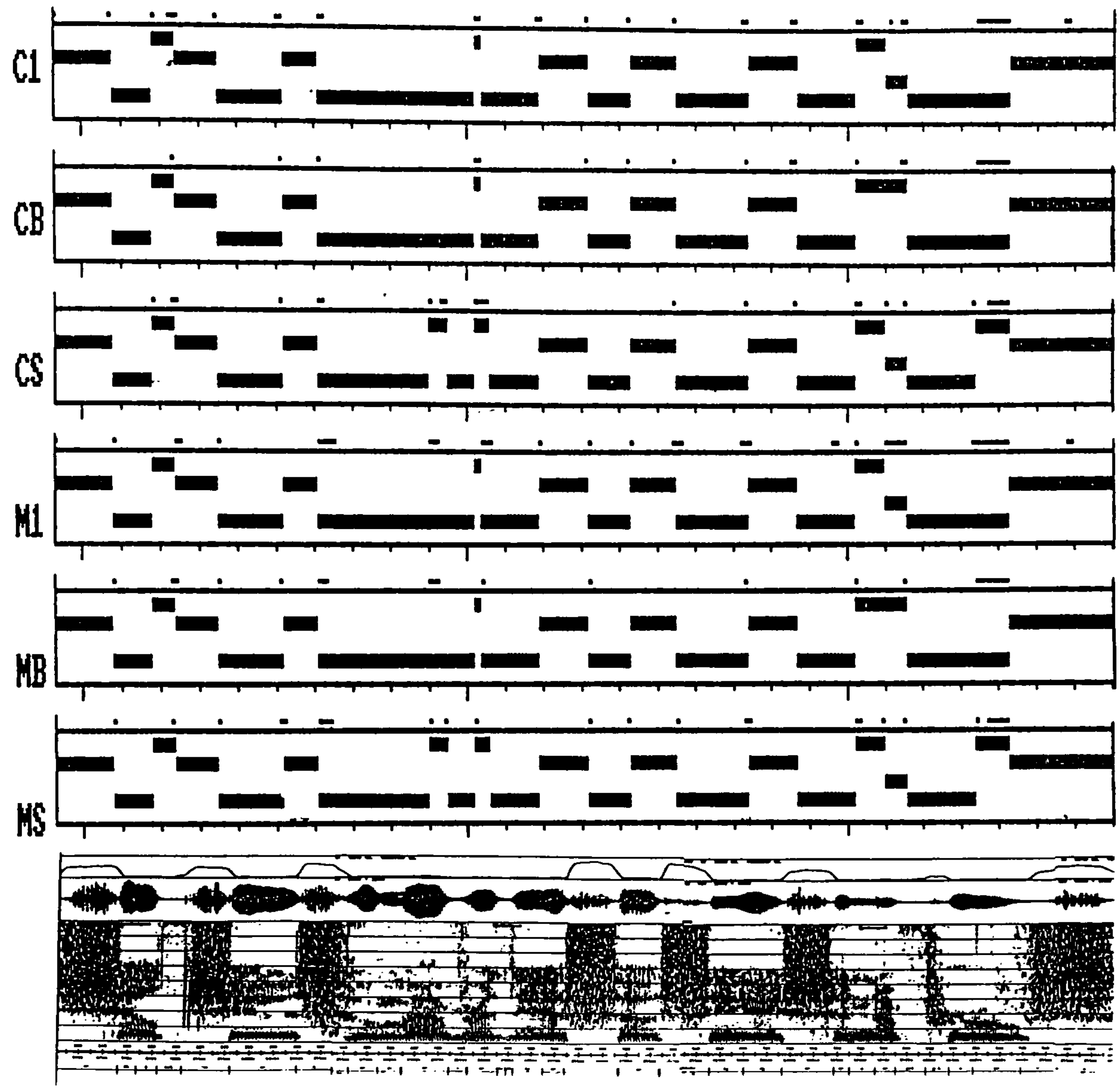


Figure 3.7 Classification results on utterance:
"She'll choose whomever the association approves."

Note:

1 Each partition, denoted by the time tick marks along the bottom of the graph, represents 100 msecs, or 10 frames.

2 The thick bar indicates the true coarse class.

3 The thin line along the top of each graph indicates where the classifier made an incorrect classification.

4 The lowest level bar is silence, next up is vowel-like, then weak fricatives, and finally strong fricatives.

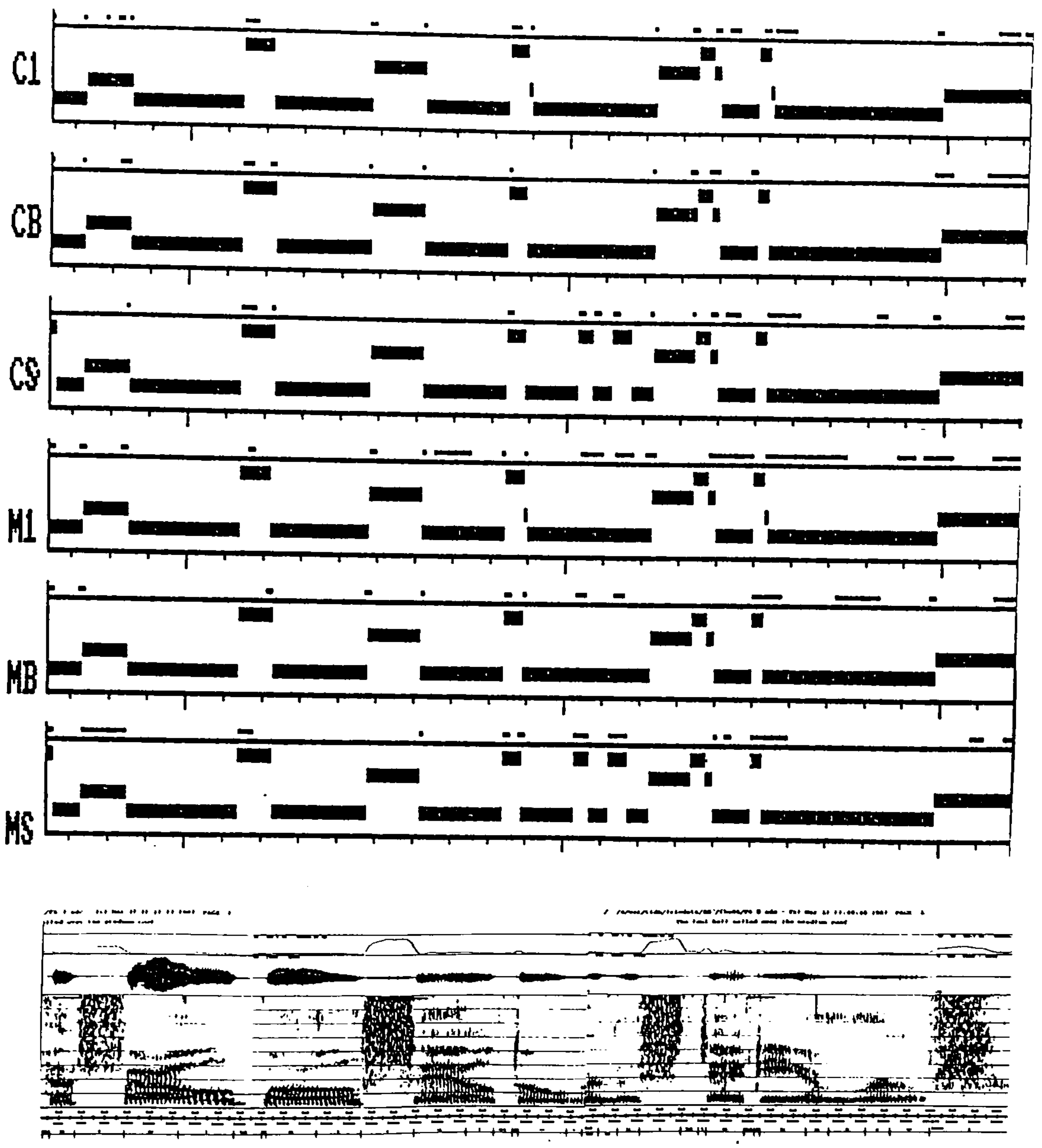


Figure 3.8 Classification results on utterance: "The foul ball sailed over the stadium roof."

CHAPTER 4

CONCLUSIONS AND FURTHER STUDY

The previous chapters of this paper have presented the methods and results of a comparison study of six different coarse phonetic classifiers. This chapter will draw some conclusions from the results and make some suggestions for future work.

4.1 Tree structure influence

One of the goals of this study was to examine the influence of tree structure on the performance of the classification process. The results showed that the overall performance of the classifier was improved by the use of a tree structure as opposed to a single decision structure. From Figure 3.6 it can be seen that for the K-means method the tree structure increased the performance by from one to nine percent across the performance figures. For maximum likelihood, the improvement was even greater, giving increases ranging from zero up to twelve percent. All three of the top performers (upper half) were tree structured. Of all the structures, the two binary tree (MB and KB) finished first and second, indicating that this structure has an inherent advantage. Another influence of the tree structure is the ability to allow the system designer to improve recognition of one particular class at the expense of the others. This is particularly true of the skewed binary tree structure. Here, the system designer may place the class needing the highest recognition rate at the first terminal node in the tree. In this study, the strong fricatives were placed in that position, and the recognition rate went from 74% in the single level tree to 89% in the skewed tree using maximum likelihood, see Appendix D, Table D.1.

4.2 K-means versus Maximum Likelihood

When comparing the results of K-means and maximum likelihood methods, there are two areas to be considered. First, the overall ability to work with coarse classes, and, second, the best performance using the optimal feature sets for each class.

In overall ability to be trained on different feature sets, the maximum likelihood method did not perform well. Depending on the set of data that was used (i.e. all males, one speaker, all speakers), the training process failed to generate covariance matrices on most combinations of features. This is attributed to the fact that this method is much more sensitive to the distribution of the data than K-means. In the K-means method, any combination of features always worked with any set of speakers. This is because although K-means assumes that the data are normally distributed, there are no calculations that actually rely on that being true. Therefore, even with non-normally distributed data, the K-means process functions; however, it may return poor results.

The second results comparison is the classification performance using the best set of features for each method. Here, the two methods performed almost the same, coming within one percent of each other in three of the four performance measures shown in Figure 3.6. K-means, using a simple Euclidian distance measure, took the second and third overall positions. All of the top three classifiers were in fact very close in their performances. It is suspected that one reason for the good performance of the simple Euclidian distance is that the very classes the system was trained on were determined using that same distance measure. If the maximum likelihood measure were incorporated into a clustering algorithm, and then used to determine the classes, it may have then shown a more substantial improvement over K-means. Another reason that the K-means method performed as well as it did is that maximum likelihood had significant trouble generating covariance matrices during training. It is suspected that by removing some of the outlying data points from the training set, we could allow maximum likelihood to further improve its performance. The creation of this training set would be a very interesting extension of this work.

4.3 Normal Distribution

Both the K-means and maximum likelihood make the assumption that the vectors within each coarse class form a normal distribution. The results of testing indicate that this is not the case. The set of data appears to be made up of many "hills and valleys," even within a single coarse class. When K-means is run in its unconstrained form on truly normally distributed data, it migrates toward the peaks and stabilizes there, letting the algorithm terminate. When it was run on a large set of data for this study, the centers drifted from one small peak to another. Eventually, the system allowed one center to hold almost all of the data points while the remaining centers were placed on some insignificant peaks. This indicates that there are too many small peaks in the data and no clear center point within the coarse classes. With the maximum likelihood method, there was significant difficulty generating covariance matrices which also indicates non-normally distributed data. Out of the 1024 combinations of features, only five were successful in generating covariance matrices capable of separating all four classes.

It is clear that either the classification algorithm used must be designed for non-normally distributed data, or the data must be reorganized to allow it to be viewed as normally distributed. In a recent broad classification study at Carnegie Mellon by Chigier and Brennan [CHIG88], it was found that within their three broad classes there were 19 subclasses that were each more normally distributed than the broad classes (see Figure 4.1). To determine the probability of a sample being in a broad class, they took the sum of the probabilities that the sample was in each of the subclasses. This is a very attractive approach that might be exploited to improve the results of this work.

Class	Description	Broad class
background	Long noisy background (inhalations etc)	Silence
noisybackground	Long non-noisy background and silences	Silence
closures	Clean voiceless closures	Silence
noisyclosures	Noisy closures	Silence
voicedclosure	Voiced closures	Silence
unvoicedv	The unvoiced allophone of v	Stop-fricative
hv	The phoneme hv	Stop-fricative
strongfricative	Strong fricatives (e.g., s, sh, z)	Stop-fricative
affricates	Affricates (ch, jh)	Stop-fricative
weakfric	Weak fricatives (e.g., f, th)	Stop-fricative
strongptk	Heavily aspirated stops (e.g., p, t, k)	Stop-fricative
weakptk	Weakly aspirated stops	Stop-fricative
voicedstops	Voiced stops (e.g., b, d, g)	Stop-fricative
nasal	Nasals (e.g., n, m)	Sonorant
voicedv	The voiced allophone of v	Sonorant
liquids	Liquids (e.g., l, r, w)	Sonorant
backvowels	Back vowels (e.g., ao, uw)	Sonorant
frontvowels	Front vowels (e.g., iy, ey)	Sonorant
midvowels	Mid vowels (e.g., ax, oe)	Sonorant

Figure 4.1 The 19 subclasses within 3 broad classes as determined by a coarse classification study at Carnegie Mellon, done by Chigier and Brennan.

4.4 Classification probabilities

Another consideration related to the non-normal distribution problem is the usefulness of the probability. The classifiers return a probability based on the distance between an unknown frame's vector and the center of the coarse class to which it has been assigned. These probabilities are difficult to use and, in fact, can be misleading. For example, consider the class of vowels. Given a representative sample set of vowels, they will tend to be distributed among the three classes: back vowels, mid vowels, and front vowels. When the coarse class of all vowels is constructed, it will determine a center point somewhere in the middle of the triangle created by these three subclasses. Now, when the system receives a frame that is the phoneme /ao/ to classify, rather than falling right on or very close to the center point for vowels, it will be away from the center, nearer to the back vowels. For this reason, speech samples that are very typical and would be expected to receive a very high probability, often receive a low probability. Therefore, in order to take advantage of the probability, the system must have a more accurate scheme for determining the probability. If the system were to incorporate the concept of subclasses within a coarse class, then once the coarse class had been determined, it could return the highest probability of any subclass within the coarse class as the probability for the frame.

4.5 Smoothing the results

The current system operates strictly on a frame-by-frame basis, taking almost no account of the characteristics of its neighboring frames. The only constraint involving neighbors is if a frame is surrounded on both sides by frames classified differently than itself, than the frame's class will be changed. A possible extension of the present work would involve designing a second level of software to smooth the classes. For example, it was observed that the first 20 or 30 msec of a strong fricative is sometimes classified as a weak fricative, while the remaining 60 to 100 msec is classified correctly. This might be detected and corrected by a program that examines segments larger than 10 msec. Another example might be the first 20 to 30 msec of a /t/ that looks very much like an /s/ could be detected by the silent period preceding it. If a vowel-like segment under 50 msec surrounded on both sides by some other class were detected, it would be reasonable to merge the vowel-like segment into the adjacent segments. This type of process can be done using a rule-based system trained by observation. The Hidden Markov Model technique [RABI86] would be a very attractive method to perform this function. It could train itself using the output of the first-choice outputs of the classifier along with the true classes for each frame. The goal of the training phase would be to find the most common errors and determine the best way to correct them.

4.6 Summary

The task of coarse classification and segmentation can be a very useful first phase in a phonetically-based speech recognition system. This study has compared the results of six coarse classifiers using combinations of the K-means algorithm, maximum likelihood, and several different tree-structured decision processes. It was found that the maximum likelihood method, using a binary tree structure, performed coarse classification more accurately than any other combination. Of the 26,000 frames tested, 80% were correctly classified under this structure. Error analysis indicated that a substantial portion of the errors occurred at the edges of the coarse segments. It is felt that the type of errors occurring could be greatly reduced by higher level software.

In addition to a frame-by-frame analysis, the results were analyzed on a coarse segment basis. All frames within each segment were labeled correctly (within 10 msec of the edge) for about 58% of the segments. Over 90% of the segments had at least some frames within them labeled correctly. Some of the types of errors occurring might be detected and corrected by a post processor looking at several frames at a time. It was also found that the data within the coarse classes are not normally distributed, which caused considerable difficulty during the training process using the maximum likelihood technique. However, it is felt that by further breaking down the classes and ignoring some of the atypical samples during training, even higher performance rates can be achieved. The results have been encouraging, and it is expected that future work will be able to use the tree structure approach outlined here in other areas of phonetic classification.

CHAPTER 5
USER DOCUMENTATION

5.1 Introduction to the CLASS facilities

The CLASS system consists of a set of programs designed for the purpose of training and classifying speech utterances into coarse phonetic classes. In this chapter, each of the programs will be discussed in detail concerning their functionality, input, and output. Without a complete understanding of this thesis, it is intended that from this chapter the user could perform training and classification experiments on a variety of tree structures, speakers, features, and classes. It is expected that the user has a basic understanding of phonetics, the features being extracted, and the tree structured decision making process.

The programs written for CLASS each build on the output of the previous program. This collective knowledge is accumulated in a "statistics" file that is complete by the end of the training process. The statistics file is then used to classify an unknown utterance. Because this file is added to in increments, it is important the programs be run in the order in which they are specified. The remainder of this chapter is organized as follows:

- A brief description of the programs (see below)
- An overview of the program flow (Figure 5.1)
- Detailed training process explanation
- Detailed classification process explanation
- A detailed discussion of each program

PRE	This program is responsible for extracting all of the feature vectors from the training data base. It is the first program run for training both maximum likelihood and K-means.
NORM	This program will normalize the feature vectors produced by PRE. Normalization is only done for the K-means training method.
KTRAIN	This program takes the normalized feature vectors and performs the appropriate training procedures on them to produce cluster centers and standard deviations at each point in the tree structure. The training process used in this program is K-means.
CLASS	This program uses the K-means training results produced by the previous three programs and performs coarse classification on an unknown speech utterance.
SD	This program takes feature vectors and analyzes the data for center points, deviations, and feature covariance at each decision point in the tree. It also produces a file indicating the class to which each feature vector is closest. This program is run for both training and classification using the maximum likelihood method.
CLASSM	This program takes the results from the SD program and produces output files the same as those produced by CLASS. It also produces confusion matrices and error statistics. The program does not perform training or classification itself, it only reorganized the results of SD to be more easily compared to the output of CLASS.

```
             PROGRAM EXECUTION FLOW

K-means                               Maximum
                                     Likelihood
-----------------------------------------------
                        |
                        |
            Create an options file
                        |
                        |
                    RUN PRE
                     /    \
                    /      \
                   /        \
                  /          \
                 |            |
             RUN NORM         |
                 |            |
                 |       SD COM FILE
                 |            |
             RUN KTRAIN       |              Training
                 |            |
- - - - - - - - -+- - - - - - +- - - - - - - - - - - - -
                 |            |
                 |       SD COM FILE      Classification
                 |            |
             RUN CLASS        |
                 |            |
                 |        RUN CLASSM
                 |            |
                  \          /
                   \        /
                    \      /
                     \    /
                  output coarse
              phonetic transcription
```

Figure 5.1 Programs used and flow for training and classifying utterances.

5.2 Detailed discussion of the training process

In this section, each of the steps required to train a system is discussed, including the calling sequence of the programs, the input files, the output files, and any other pertinent details. The first and most important part of the training process is to generate an options file that accurately portrays the intended training parameters.

The options file is a text file that defines all of the options to all of the programs during the training process, see Appendix C for an example. The options file guides the system during the training phase. The same file will also be used during classification where only items such as the input utterance would be changed. This file must be well thought out before any reasonable results can be expected from the classifier. Once in place, this file can be changed and the system retrained and tested to evaluate the effects of the change on the accuracy of the system. These changes include using different sets of features at the decision points, using different feature parameters, and changing the tree structure.

There are five general areas to consider when creating the options file. The remainder of this section examines each of these areas in detail. The syntax for the options is given in Appendix A.

1) The training method to be employed.

Option line:

```
   TRAINING_METHOD =          ( TRAINING_METHOD = K_MEANS )
```

The training method can be set to either K-means, maximum likelihood, or fixed classes. If K-means or fixed classes is used, then the program PRE should be run, followed by NORM and KTRAIN. If maximum likelihood is used, then after running PRE, a *.unix (*.com for VMS) script file will be generated. This file will run the SD program in the appropriate manner to perform the remainder of the training necessary. The distinction between K-means and fixed classes is that the training process can either determine its own classes using the K-means clustering procedure, or it can use the classes as specified by the user in the options file. With K-means specified, the system will determine what phonemes are in each class by determining the best cluster separation, based on the feature vectors. With fixed classes, the user determines the final clusters, and the training session simply calculates the means and deviations based those classes.

2) The source files to be used.

Option line:

```
   SOURCE =              ( SOURCE = data/obey )
   PTLOLA =              ( PTLOLA = .ptl )
   FRAME_SIZE =          ( FRAME_SIZE = 5 )
```

Each occurrence of the "SOURCE" line specifies a common root name (with path if necessary) of a set of three files. These three file are the source of the speech utterance and there may be as many source file line as desired. The three source files are: first, the ".b" file that contains the PCM speech samples; secondly, the ".f" file that contains the fft's taken once per msec.; and thirdly, the ".ptlola" file which is the phonetic transcription information. The ".b" file is derived from the CMU data base ".adc". The CMU files have a different sampling rate than is used in this work; and, therefore, the speech utterance must be low pass filtered and resampled. The ".f" file is generated from the ".b" file by an fft program called "b2f". The ".ptlola" file is taken unchanged from the CMU data base. Because this project runs on several computers, some of which do not support file name extensions greater than three characters, the "PTLOLA=" option was introduced. The character string specified here will be used for the file extension of the phonetic label file. FRAME_SIZE specifies the number of msec.'s in a frame. The default here is 10 msec.

3) The features to be used.

```
Option line:
   FEATURE =                  ( FEATURE = ZERO_CROSSING,-200,200 )
   FEATURE_FILE =             ( FEATURE_FILE = OBEY )
   STANDARD_DEVIATION =       ( STANDARD_DEVIATION = 10000 )
```

In this section, we must consider the features that will be used. The line "FEATURE=" specifies the features that are to be extracted from the source files. As the features are being extracted, their values are placed into files named with a root as specified on the "FEATURE_FILE=" line and an extension corresponding to the feature. Each feature is placed in a separate file. The "STANDARD_DEVIATION=" line indicates what the standard deviation should be when the features are normalized during preparation for the K-means clustering.

4) Phoneme selection and seed classification.

```
Option line:
   PHONEME =           ( PHONEME = ow,b,y = 5 )
   CLASS =             ( CLASS = 0,ow,y )
```

These two options give the user control over the phonemes selected for the training process and the initial classes. The "PHONEME=" line can specify a phoneme (or several phonemes), followed by a count. The count will limit to that number the occurrences of the phoneme specified. If a count is not specified, then all of those phonemes found in the source samples will be used. This count should be used if there is a particular phoneme, or several phonemes that occur so frequently that they dominate the training set. They then can be limited to a reasonable number. The "CLASS=" option allows the user to specify which phonemes belong to each class. By using these options together, the user can, for example, only extract vowels and set the classes 0, 1, and 2 to the three broad vowel classes. In this way, the system could be trained and tested on the usefulness of each of the features in distinguishing between the three broad vowel categories.

5) The decision tree structure to be used.

Option line:

```
   NODE =             ( NODE = 0.1 = 2,C0 )
```

A sequence of these lines defines the tree structure. The tree can be of virtually any shape and size. The node lines MUST appear in the order of a breadth-first, right-to-left tree search. The nodes are internally numbered starting with zero in the order that they appear in this list. Each node line specifies either the number of children (if it is defining a non-terminal node) or the class that the node represents (if it is a terminal node). If the training method specifies "FIXED_CLASSES," then these will represent the final classes from the training process. If this is not specified, the final classes may be different, and these are only used to find seed vectors for the k-train clustering algorithm. On a non-terminal node, the features to be used can be specified. If features are declared optional, the training process then tries the feature set with all combinations of the optional features. It chooses the best feature set based on the least number of incorrectly placed frames.

5.3 Detailed discussion of the classification process

The classification process for the K-means system involves simply running the program CLASS. There are three typical ways of classifying an unknown utterance. First if the classification is to be done on the same utterance set as the training was performed, then CLASS can be run passing it only the options file. Second, if a new utterance is to be used it can be classified by running CLASS and passing it the options file (unchanged from the training process) and the root name of the files containing the utterance to be classified. The files containing the utterance must include a .b file, a .f file and an optional .ptlola file. If the .ptlola file is missing then error analysis results will not be generated. The third way to classify utterances is by modifying the options file to have the SOURCE line reference utterances other than the ones that were trained on, then running CLASS passing it only the options file. This would have to be done if several utterances were to be classified together.

The classification process for maximum likelihood requires the following four steps:

1) running PRE on the new utterance[s]
2) making a modification to the .unix (.com) file
3) running that new .unix (.com) file
4) running CLASSM

The first step is to run PRE on the utterances to be classified. If the same utterances are to be used as were used during training, then steps 1 through 3 can be skipped. If new utterances are to be classified, then the options file needs to be modified. The SOURCE line of the options file needs to include the file names of the new utterances. PRE is then run with the modified options file. This will produce the appropriate .sxx (vector data) files to be used by the .unix file. Next the .unix file must be modified to use the trained statistics file as input on the "-s" switch. Also, all calls to SD should use the .s00 file generated by PRE. There are comments in the .unix file that will more clearly state what changes need to be made. Once these changes are made the .unix file may be run. After completion of the .unix file the CLASSM program should be run. This program takes the output files from SD and the original hand label information files and displays confusion matrices and produces files in the same format as CLASS.

5.4 Detailed discussion of the each program

In this section each of the programs shown in Figure 5.1 are discussed with a functional overview and a description of usage, input, and output.

5.4.1 PRE - The feature extraction program

The feature extraction program PRE is a preprocessor to the training routines. Its main function is to extract all the features requested from all of the source files specified. The input to this program includes: the FFT file (.f), the PCM file (.b), the hand label file (.ptlola), and the options file (.opt). PRE produces a set of files that contains the extracted features. There will be a file for each feature, and all files will contain the same number of 16-bit integers. Taking the first integer from each of the files and putting them together gives the feature vector for the first frame, the second integer from each file will return the second feature vector, and so on. PRE also produces a file containing the coarse phonetic labels for each sample. This, too, is a file of 16-bit integers, where each word represents a phoneme. This file is used by the training process to determine how well each set of features performed in the task of splitting the phonemes into coarse classes. Finally, PRE will start the process of building up the statistics file. This is a file that contains information about the training process and is the input file used by the classifier.

Usage:

```
pre <opt file> [-v] [-d]
```

where:

<opt file>	is the root file name of the options file. The extension will be added as .opt.
-v	Verbose output. This returns some additional information to the screen when the program is running.
-d	Debug output. This returns more information to the screen when the program is running.

Input:

.opt	This is the options file and must be specified as the first parameter on the command line when calling PRE.
.b	The PCM samples are stored in files named with the extension ".b". The files to be used are specified in the options file.
.f	For each .b file there is expected to be a corresponding .f file that contains the FFT data. This file must be in the same directory and have the same root name as the .b file.
.ptlola	For each .b file, there is expected to be a corresponding .ptlola file that contains the hand-labeled phonetic translation of the speech utterance. This file must be in the same directory and have the same root name as the .b file.

Output:

All output files will have their root names as specified in the options file on the "FEATURE_FILE=" line.

When run with "METHOD=K_MEANS":

.zer	Zero crossing rate feature data points.
.tot	Total energy feature data points.
.rel	Relative energy feature data points.
.pea	Peak energy feature data points.
.spe	Spectral change feature data points.
.per	Periodicity feature data points.
.mo1	First moment, mean, feature data points.
.mo2	Second moment, deviation, feature data points.
.mo3	Third moment, skew, feature data points.
.mo4	Fourth moment, kurtosis, feature data points.
.pt	Phonetic label for each data point as specified by the hand label file.

When run with "METHOD=MAX_LIKELIHOOD":

.sxx	where xx represents the node number, and the file contains all of the feature vectors for that decision point. This file is in a format to be sent directly to the SD program.
.unix	This is a Unix script file that is used to perform the training. It is generated specifically for Unix, and the mode will have to be changed to add execution privileges.
.com	This is a VMS command file that is used to perform the training on a VAX running VMS.
.pt	Phonetic label for each data point as specified by the hand label file.

5.4.2 NORM - The data normalizer

Once the feature vectors have been collected, it is important to be sure that no one feature dominates the decision making process. In order to insure this, the features are normalized. The normalization process merely involves finding two numbers X and Y such that when all of the occurrences of a features are modified by X * (point + Y), the standard deviation will be that specified in the options file and the mean will be at zero. This process is done for each of the features independently.

Usage:

```
        norm <opt file> [-v] [-d]

        where:
                <opt file>   is the root file name of the options
                             file.  The extension will be added as
                             .opt.

                -v           Verbose output.  This returns some
                             additional information to the screen
                             when the program is running.

                -d           Debug output.  This returns more
                             information to the screen when the
                             program is running.
```

Input:
All input files are expected to have their root names as specified in the options file on the "FEATURE_FILE=" line.

.opt	This is the options file and must be specified as the first parameter on the command line when calling NORM.
.zer	Non-normalized zero crossing rate feature data points.
.tot	Non-normalized total energy feature data points.
.rel	Non-normalized relative energy feature data points.
.pea	Non-normalized peak energy feature data points.
.spe	Non-normalized spectral change feature data points.
.per	Non-normalized periodicity feature data points.
.mo1	Non-normalized first moment, mean, feature data points.
.mo2	Non-normalized second moment, deviation, feature data points.
.mo3	Non-normalized third moment, skew, feature data points.
.mo4	Non-normalized forth moment, kurtosis, feature data points.

Output:
All output files will have their root names as specified in the options file on the "FEATURE_FILE=" line.

.zer	Normalized zero crossing rate feature data points.
.tot	Normalized total energy feature data points.
.rel	Normalized relative energy feature data points.
.pea	Normalized peak energy feature data points.
.spe	Normalized spectral change feature data points.
.per	Normalized periodicity feature data points.
.mo1	Normalized first moment, mean, feature data points.
.mo2	Normalized second moment, deviation, feature data points.
.mo3	Normalized third moment, skew, feature data points.
.mo4	Normalized forth moment, kurtosis, feature data points.
.sta	Statistics file.

5.4.3 KTRAIN - K-means training program

The main task of KTRAIN is to produce a mean and standard deviation for each node in the decision tree. This is done by associating all of the feature vectors in the root node of the tree and then separating the vectors into as many clusters as there are children off of the root node, (see Figure 2.5). Once this is done, the mean vector and standard deviation of each cluster is determined and written to the statistics file. Then the process is repeated on each of the children. When all of the vectors are in terminal nodes, the process stops, and the training is considered complete. Two major considerations must be addressed when running KTRAIN. The first is the algorithm used in dividing the vectors associated with a node among its children. This can be done by splitting the vectors so that they move toward their declared terminal nodes. No intelligence is needed the part of the program with respect to clustering the data in this case. If the classes are not known then the K-means training algorithm may be employed (see section 1.5). In this case the feature vectors themselves determine how they are grouped based on their relative distances from each other. The second major consideration is what features should be used at each decision point in the tree. The features desired are declared on the "NODE=" line for each decision point.

```
Usage:
        ktrain <opt file> [-v] [-d]

        where:
               <opt file>  is the root file name of the options
                           file. The extension will be added
                           as .opt.
```

-v	Verbose output. This returns some additional information to the screen when the program is running.
-d	Debug output. This returns more information to the screen when the program is running.

Input:

All input files are expected to have their root names as specified in the options file on the "FEATURE_FILE=" line.

.opt	This is the options file and must be specified as the first parameter on the command line when calling NORM.
.zer	Zero crossing rate feature data points.
.tot	Total energy feature data points.
.rel	Relative energy feature data points.
.pea	Peak energy feature data points.
.spe	Spectral change feature data points.
.per	Periodicity feature data points.
.mo1	First moment, mean, feature data points.
.mo2	Second moment, deviation, feature data points.
.mo3	Third moment, skew, feature data points.
.mo4	Fourth moment, kurtosis, feature data points.
.pt	Phonetic label for each data point as specified by the hand label file.
.sta	Statistics file.

Output:

The output file will have the root name as specified in the options file on the "FEATURE_FILE=" line.

.sta	Statistics file.

5.4.4 CLASS - The classifier

This program utilizes the results of the training process for K-means. It will read in the statistics file and an unknown utterance then perform coarse classification on that utterance. On the screen a frame by frame indication of the classification process came be seen (using -v) and confusion matrices will be displayed at the completion of the classification process.

Usage:

```
        class <opt file>  [ <utt file> ]  [-v]  [-d]

        where:
           <opt file>   is the root file name of the options
                        file.  The default extension is .opt.

           <utt file>   Speech utterance file name.  This should
                        be the path and root file name of the
                        pair of files, including a .b file and
                        a .f file.  If the .ptlola file is also
                        available, it will be read, and performance
                        indicators will be generated.  If not
                        specified the source files from the .opt
                        file will be used.

           -v           Verbose output.  This returns some
                        additional information to the screen
                        when the program is running.

           -d           Debug output.  This returns more
                        information to the screen when the
                        program is running.
```

Input:
All input files are expected to have their root names as specified in the options file on the "FEATURE_FILE=" line.

Output:
The output file will have the root name as specified in the options file on the "FEATURE_FILE=" line.

```
       .tru     True class for each frame
       .1st     First choice class for each frame
       .2nd     First choice class for each frame
       .err     Error indication per frame:
                  0=no error
                  1=missed first choice
                  2=missed first and second choice
```

5.4.5 SD - Maximum likelihood training program

This program was developed for another speech research project and is used here to determine its effectiveness with respect to the K-means clustering approach. This program uses the maximum likelihood formula (see section 1.6), which incorporates feature covariance into the distance between vectors. Because it was not written with the tree structure in mind, it must be run once for each decision point in the tree rather than once for the entire tree. To make this a bit easier, PRE produces a script that runs the program the appropriate number of times with the appropriate options. This will be generated whenever the training method is specified as "MAX_LIKELIHOOD" in the options file. For a detailed discussion of the usage and working of the SD program refer to [GAYV88]. For the purpose of this work, it should be sufficient to run the script file produced by PRE.

Usage:

```
        test_file.com

                This is the script file produced by PRE.
                Execution privileges may need to be added.
```

Input:

.sxx	Feature files as produced by PRE.

Output:

.s	Statistics file.

5.4.6 CLASSM - Classifier for Maximum Likelihood

This program gathers together the files produced by the several runs of SD into the same set of files as the program CLASS produces. This program produces confusion matrices. However, because it is not actually doing any classification it will not use the options file.

```
Usage:
      classm <tree type> <root file>

      where:
            <tree type> indicates the tree structure being
                        used and must be one of the following:
                            1  - single level tree
                            b  - binary tree
                            s  - skewed binary tree

Input:
   All input files are expected to contain the results of the
   decisions made by the SD program at each decision point in
   the tree structure.

   .pt      File containing the true phonetic labels

            If 1 is specified:
   _0.1st   First choice at decision node 0.

            If b is specified:
   _0.1st   First choice at decision node 0.
   _1.1st   First choice at decision node 1.
   _2.1st   First choice at decision node 2.

            If s is specified:
   _0.1st   First choice at decision node 0.
   _1.1st   First choice at decision node 1.
   _3.1st   First choice at decision node 3.


Output:
   All output files will have their root names as specified in the
   second parameter passed to the program.

       .tru     True class for each frame
       .1st     First choice class for each frame
       .2nd     First choice class for each frame
       .err     Error indication per frame:
                  0=no error
                  1=missed first choice
                  2=missed first and second choice
```

CHAPTER 6

REFERENCES

[CARL68] Carlson Communication Systems: An Introduction to Signals and Noise in Electrical Communication, McGraw-Hill Book Company, 1968.

[CHEN86] Chen, F. "Lexical access and verification in a broad phonetic approach to continuous digit recognition", Proceeding of the IEEE ICASSP 1986, 1089-92.

[CHIG88] Chigier, B. and Brennan, R., "Broad class network generation using a combination of rules and statistics for speaker independent continuous speech", Proceeding of the IEEE ICASSP 1988, 449-52.

[COLE80] Cole, R. and Zue, V., "Speech as eyes see it," In Attention and Performance VIII, R.S. Nickerson, Ed. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1980, 475-494.

[COLE86] Cole, R., Phillips, M., Brennan, B., and Chigier, B., "The CMU phonetic classification system", Proceedings of the IEEE ICASSP 1986, 2255-57.

[DUDA73] Duda, R. and Hart, P., Pattern Classification and Scene Analysis, John Wiley and Sons, 1973.

[FORR88] Forrest, K., Weismer, G., Milenkovic, P. and Dougall, R., "Statistical analysis of word-initial voiceless obstruents", University of Wisconsin, 1988.

[GAYV88] Gayvert, R., "Statistical methods for formant tracking", Masters thesis, Rochester Institute of Technology, Rochester, New York, 1988.

[GLAS84] Glass, J., "Nasal consonants and nasalized vowels: An acoustic study and recognition experiment", Masters thesis at Massachusetts Institute of Technology, December 1984.

[GLAS85] Glass, J. and Zue, V., "Detection of nasalized vowels in American English", IEEE 1985 Acoustics, Speech, and Signal Processing, 1569-1572.

[GLAS87] Glass, J. and Zue, V., "Acoustic segmentation and classification", Report presented at the DARPA program review, San Diego, CA, March 24-26, 1987.

[LEUN85] Leung H. and Zue, V., "Automatic alignment of phonetic transcription with continuous speech", Proceedings of the IASTED International Symposium on Robotics and Automation, Lugano, Switzerland, June 24-26, 1985.

[MAEN84] Maenobu, Ariki, Sakai, "Speaker-independent word recognition in connected speech on the basis of phoneme recognition", Information Sciences 33, 1984, 31-61.

[MEIS86] Meisel, W., "Implications of large vocabulary recognition," Proceeding of the IEEE Speech Tech "86, April 1986, 189-192.

[RABI86] Rabiner, L. and Juang, B., "An Introduction to Hidden Markov Models", IEEE ASSP Magazine, January 1986, 4-16.

[SMIT86] Smith, B., Hillenbrand J., and Ingrisano D., "A comparison of temporal measures of speech using spectrograms and digital oscillograms" Journal of Speech and Hearing Research", 29, 1986, 270-274.

[TOU 74] Tou, J. and Gonzalez, R., Pattern Recognition Principles, Addison-Wesley Publishing Comp., 1974

[WILC86] Wilcox, L., and Lowerre, B., "Coarse classification using a hierarchical decision tree and top down parsing", Hewlett-Packard Laboratories, Proceeding of the IEEE ICASSP 1986, 73-76.

[WILP85] Wilpon, J. and Rabiner, L., "A modified K-means clustering algorithm for use in isolated word recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-33, 1985, 587-93.

[ZUE 85] Zue, V., "The use of speech knowledge in automatic speech recognition", Proceedings of the IEEE, 73, 1985, 1602-1615

[ZUE 79] Zue, V. and Cole, R., "Experiments on spectrogram reading," Proceedings of the IEEE ICASSP-79, 1979, 116-119.

[ZUE 82] Zue, V. and Schwartz, "Acoustic processing and phonetic analysis", Trends in Speech Recognition, 101-124.

CHAPTER 7

GLOSSARY

1) Children - all nodes directly connected to and one level below a given node are children of the given node.

2) Coarticulation - the process in which a phoneme is influenced by adjacent phonemes during continuous speech.

3) Euclidian distance - a linear distance measure between two points in an n-dimensional space. It is calculated by taking the square root of the sum of the squared distances for each dimension.

4) Formant - a resonance frequency of the vocal tract.

5) Frame - a fixed time slice (e.g. 10 msec) of speech utterance.

6) PCM - Pulse Code Modulation - digital modulation in which a message is represented by a coded group of digital (discrete-amplitude) pulses [CARL68].

7) Phoneme - the smallest unit of speech that distinguishes one utterance from another. It displays variation in the speech of a single person or in a particular dialect as the result of modifying influences. Definition from Webster's Third New International Dictionary.

8) Phonetic alphabet - a set of symbols used in phonetic transcription. There is a separate symbol for every speech sound that can be distinguished.

9) RMS energy - Root Mean Squared - a calculation of energy.

10) Segment - a continuous time slice of the speech utterance where the entire slice is all of the same coarse class, (i.e. vowel, strong fricative).

11) Segmentation - the process of taking in continuous voice signal and dividing it into phonetic segments.

12) Spectrogram - a two dimensional representation of speech, showing frequency on the Y-axis, time on the X-axis and intensity using light to dark shading.

13) Terminal node - a node in a tree structure with no branches emerging from it.

APPENDIX A

A OPTION FILE FORMAT

This appendix explains the format of the options file used throughout the training and classification process. The option file is an ASCII text file that allows the user to control many parameters about the system being trained. The user can specify items such as the tree structure to be used, the training method, what features to use and when, etc. This file must be passed to all three steps of the training phase and also to the classifier. It is important that this file be passed to all four programs without being modified. For a detailed discussion of the programs and data flow, refer to the "User Documentation" chapter of this paper.

Comments

All text to the right of a ";" is considered comments and is not parsed as part of the options. Also, all blanks and blank lines are ignored.

METHOD = training type

This line indicates what type of training will be done. The "training type" may be one of three options: K_MEANS, MAX_LIKELIHOOD, or FIXED_CLASSES. If K_MEANS is chosen, then the k-means algorithm will be used to segment the data at each decision point in the tree during training. If FIXED_CLASSES is chosen, then the classes that were specified by the user are used to segment the data at each decision point. If MAX_LIKELIHOOD is specified, then the maximum likelihood formula will be used to make each decision in the tree.

SOURCE = file name

This is the root name of a speech utterance to be used in the training session. It is expected that there will be three files for each root name given: a ".b" file containing PCM, a ".f" file containing the FFT's, and a ".ptlola" file containing the phonetic label information. One or more of these lines can be specified. Wild card names are not supported, and the full path name should be specified for clarity.

FEATURE = feature [,p1 [,p2 [,p3 [,p4]]]]

Specifies what features are to be extracted from the speech utterance. One or more of these lines can be specified, each with one feature chosen from the following: ZERO_CROSSING, TOTAL_ENERGY, PEAK_ENERGY, RELATIVE_ENERGY, SPECTRAL_CHANGE, PERIODICITY, MOMENT_1ST, MOMENT_2ND, MOMENT_3RD, MOMENT_4TH. Several features have parameters that can be changed to maximize the usefulness of the feature. ZERO_CROSSING has a dead band associated with it through which the signal must completely pass before being counted as a crossing. This can be set by assigning the lower limit to p1 and the upper limit to p2. TOTAL_ENERGY can be limited to a specified frequency band by setting the lower limit to p1 and the upper limit to p2. RELATIVE_ENERGY is the energy in a given band relative to that of another band. The lower and upper limits of the first band are given in p1 and p2, respectively. The second band's lower and upper limits are specified by p3 and p4.

PHONEME = phonemes [= num]

Specifies which phonemes and how many of each are to be used for the training process. Each input file has a phonetic transcription file, and the feature extractor will only extract data samples from phoneme types specified here. The phonemes are CMU type names separated by commas. If a "*" is specified, then all phonemes are used. The optional second equal sign, followed by a number, indicates how many samples of the specified phonemes the feature extractor is to use. If omitted, all samples of that phoneme will be extracted. One or more of these lines may be specified. If a phoneme is specified in more than one line, the last line will determine how many are used.

FRAME = msec

Specifies the number of milliseconds (1/1000 of a second) in each frame. If not specified, the frame will be 10 msec.

SKIP = count

Skip controls how many frames are considered to be at a segment boundary. For example, if 1 is specified, then one frame on each end of a segment is considered to be an edge and is not added into the non-edge confusion matrices for the classifier. This is employed to discover the effect that removing these edge frames has on the performance of the classifier. This option will also be used by the preprocessor, and any frames on edge boundaries there will not be placed into the training set. In order to get all frames, this should be set to zero, which is the default. It is, therefore, likely that this would be set to zero for the training programs. But, during classification, it could be set to one or two. For this study, it was set to zero for training and to one during classification.

<u>NODE</u> = <u>node id</u> = <u>children</u> [, <u>p1</u>]
These lines describe the tree structure to be used. One of these lines must be present for each node in the tree. The "node id" indicates which node this line is describing (0 = root; 0.0 = leftmost child of 0; 0.1 = next child of root to right; then 0.2, 0.3, etc. 0.0.0 is leftmost child of 0.0; then 0.0.1, 0.0.2, etc.). The parameter "children" is the number of children belonging to the node. The optional parameter "p1" can take on one of two meanings depending on whether the node, is a terminal node of a decision point. If it is a terminal node this parameter will tell the system what class we want it to train this node to be (e.g. C0 for class 0). For terminal nodes, this parameter is not optional. If the node is a decision point (i.e. non-terminal), then "p1" may be a list of features that are to be used at that point. These features should be the first three letters of the features' names, each separated by commas and appearing in lower case. If an asterisk appears before the feature name, then this feature is considered to be an optional feature. The system will try all combinations of the optional features and choose the set that makes the least number of classification errors. The following is an example of a class option line with two optional features: CLASS = 1,tot,*zer,per,mol,*mo2. The feature names allowed are the following: zer, tot, pea, per, spe, rel, mol, mo2, mo3, mo4. These names are also used as the file name extensions for the feature files.

<u>CLASS</u> = <u>class number</u> [= <u>number</u>] [, <u>phoneme list</u>]
This switch is used to establish the relationship between the phonemes and the classes. The system will use initial clusters as shown in Appendix B; however, these may be rearranged to any classes desired. These are only seed classes if K-means is chosen, and the final classes may be very different. If the training method is fixed classes or maximum likelihood, then these classes will be the final classes. If the optional parameter "number" is specified, then the feature extractor will limit the number of feature vectors to the specified number. Otherwise, all feature vectors of the class will be used.

STANDARD DEVIATION = sd

Once the features have been extracted, they are normalized to achieve the standard deviation specified here. This is done to prevent features with large ranges from dominating the z-score decision during the K-means or fixed classes technique. This is not used for maximum likelihood.

INCLUDE = file name

When this line is encountered, the program will open the file specified by "file name" and begin reading lines from that file just as if they had been in the original options file. When the end of this file is encountered, processing will continue with the line following this INCLUDE line in the original options file. Only one level of include file support is allowed.

PTLOLA = file extension

This line defines the file extension used for the phonetic label file. By the conventions of the CMU data base, this is ".ptlola". However, because much of the development of this project was done on an IBM-PC, which only supports three character extensions, this switch was introduced to allow the extension to be shortened.

The following is a sample options file to be used during the training of the utterance "obey". For a complete sample of the training process using this options file, see the "user documentation" section.

```
; Options file for utterance "obey"
;
;  Tree structure:         _0_
;                        _/   \_
;                      _/       \_
;                     /           \
;                   0.0           0.1
;                  /   \          /  \
;                 /     \        /    \
;             0.0.0  0.0.1    0.1.0  0.1.1
;              sil     b        ow     y
;

METHOD = K_MEANS,FIXED_CLASSES

SOURCE = C:/CLASS/DATA/OBEY
PTLOLA = .PTL
FRAME = 10

FEATURE = ZERO_CROSSING,-400,400
FEATURE = TOTAL_ENERGY
FEATURE = RELATIVE_ENERGY,100,4000,4000,9000
FEATURE = PEAK_ENERGY
FEATURE = SPECTRAL_CHANGE
FEATURE = PERIODICITY
FEATURE = MOMENT_1ST
FEATURE = MOMENT_2ND
FEATURE = MOMENT_3RD
FEATURE = MOMENT_4TH
FEATURE_FILE = OBEY
STANDARD_DEVIATION = 1000

PHONEME = sil,ow,b,y
CLASS = 0,sil
CLASS = 1,ow
CLASS = 2,b
CLASS = 3,y

NODE = 0 = 2

NODE = 0.0 = 2
NODE = 0.1 = 2

NODE = 0.0.0 = 0,C0
NODE = 0.0.1 = 0,C2

NODE = 0.1.0 = 0,C1
NODE = 0.1.1 = 0,C3
```

APPENDIX B

B PHONEME NAMES

This appendix gives the phonetic makeup of each of the coarse classes proposed for this thesis. Following the coarse classes, a list of all of the phoneme names and their uses is presented. The class sets are not intended to be complete; they are only representative of what is being observed. The phoneme names were adopted without modification from the Carnegie-Mellon data base.

Coarse class	Typical phonemes				Phoneme groups
Silence	sil,	pau,	bg,	q,	background
	pcl,	tcl,	kcl		closures
	dcl,	bcl,	gcl		voiced closures
Strong fricatives	s,	z,	sh,	zh,	strong fricatives
	ch,	jh			affricates
Weak fricatives	f,	th,	hh,	hv,	weak fricatives
Vowels	ax,	e,	oe,	ao,	vowels
	ey,	ow,	iy,	uw	
	n,	m			nasals
	l,	r,	w		liquids

	Name	Description
Vowels:		
	iy	'beat'
	ih	'bit'
	eh	'bet'
	ae	'bat'
	ux	high, front, rounded allophone of /uw/ as in 'beauty'
	oe	mid-low, front, rounded allophone of /ow/
	ix	high, central vowel (unstressed), as in 'roses'
	ax	mid, central vowel (unstressed), as in 'the'
	ah	mid, central vowel (stressed), as in 'butt'
	uw	'boot'
	uh	'book'
	ao	'bought'
	aa	'cot'
	ey	'bait'
	ay	'bite'
	oy	'boy'
	aw	'bough'
	ow	'boat'
	e	non-diphthongized /ey/
	o	mid-low, back, non-diphthongized allophone of /ow/
Liquids:		
	l	'led'
	r	'red'
Glides:		
	y	'yet'
	w	'wet'
Syllabic resonants:		
	er	'bird'
	axr	unstressed allophone of /er/, as in 'diner'
	el	syllabic allophone of /l/, as in 'bottle'
	em	syllabic allophone of /m/, as in 'yes'em'
	en	syllabic allophone of /n/, as in 'button'
	eng	syllabic allophone of /ng/, as in 'Washington'
Stops:		
	p	'pop'
	b	'bob'
	t	'tot'
	d	'dad'
	k	'kick'
	g	'gag'
	m	'mom'
	n	'non'
	ng	'sing'
	q	glottal stop - allophone of /t/, as in Atlanta'

Affricates:

ch	'church'
jh	'judge'

Fricatives:

f	'fief'
v	'very'
th	'thief'
dh	'they'
s	'sis'
z	'zoo'
sh	'shoe'
zh	'measure'
hh	'hay'
hv	voiced allophone of /hh/, occurred between vowels

Flaps and trills:

dx	alveolar flap (allophone of /t/ and /d/)
nx	nasal flap (allophone of /n/)
lx	lateral flap (allophone of /l/)

Others:

bg	silence at beginning and end of utterance
pau	silence within an utterance
sil	same as pau, but shorter
ns	a non-speech sound
h#	exhalation at end of utterance
#h	inhalation at beginning of utterance
voi	voicing not associated with a stop closure
epi	closure resulting from coarticulation of fricative and nasal or lateral

Qualifiers:

cl	closure associated with a stop
-h	aspiration of a stop
-n	nasalization of sonorants
-q	glottalization/laryngealization of sonorants
-b	stop release at a spot where stops are not often released

APPENDIX C

C SAMPLE OUTPUT

This appendix will show the results of a sample training and classification session for the utterance "obey". The system will use four classes corresponding to the four phonemes of the utterance. Figure C.1 is the spectrogram of the utterance being used. Following this, the options file that was used is presented and then the actual output from running PRE, NORM, KTRAIN, and CLASS.

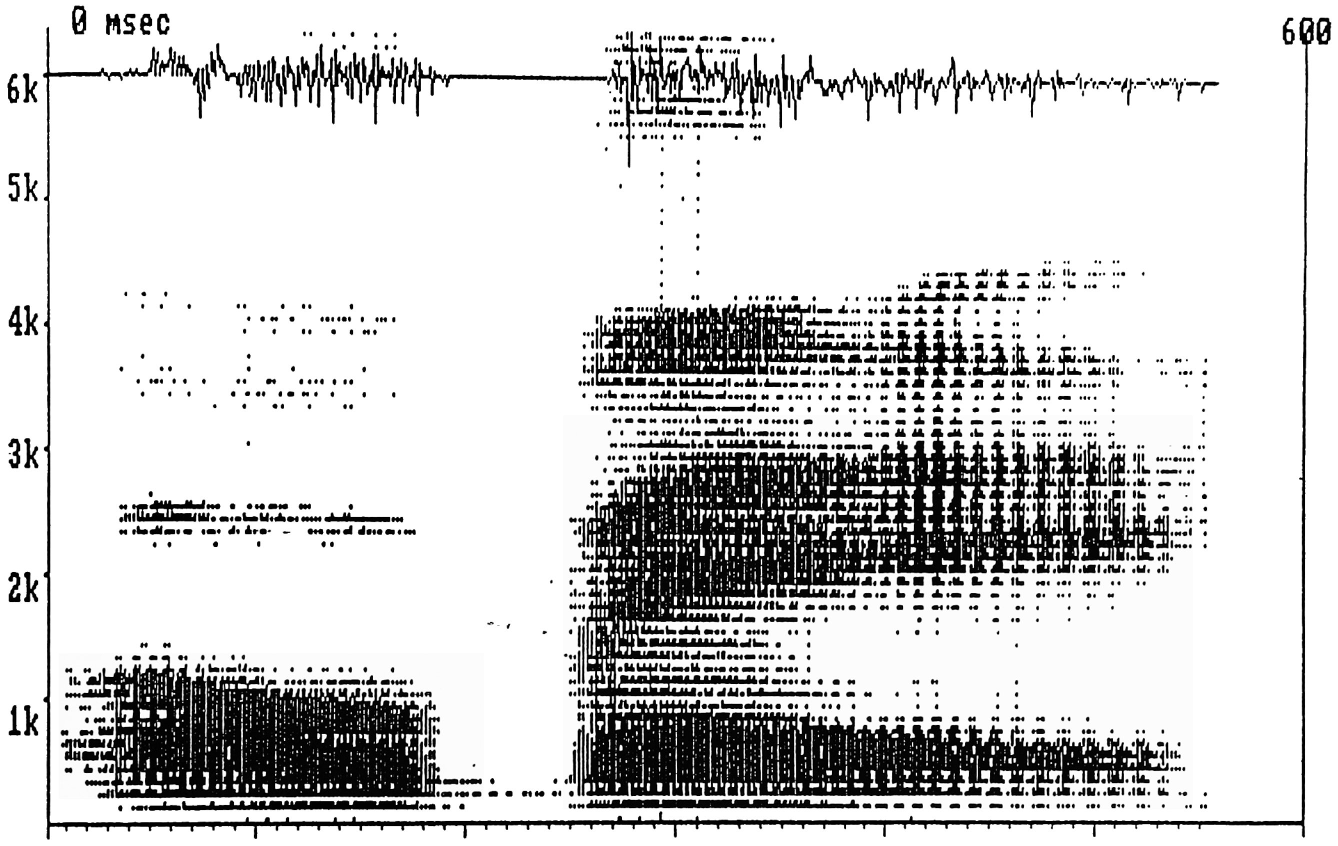


Figure C.1   Spectrogram for utterance "obey"

```
$ type obeycb.opt

;
; Options file for utterance "obey"
;
;  Tree structure:         0
;                         / \
;                        /   \
;                       /     \
;                      /       \
;                   0.0         0.1
;                   / \         / \
;                  /   \       /   \
;              0.0.0  0.0.1  0.1.0 0.1.1
;               sil     b      ow    y
;

METHOD = FIXED_CLASSES

SOURCE = [JDELMEGE.CLASS.DATA]OBEY
PTLOLA = .PTL
FRAME = 10
SKIP = 0

FEATURE_FILE = OBEY_FULL
FEATURE_SET = 1,10
STANDARD_DEVIATION = 1000
FEATURE = ZERO_CROSSING,-400,400
FEATURE = TOTAL_ENERGY
FEATURE = RELATIVE_ENERGY,100,4000,4000,9000
FEATURE = PEAK_ENERGY
FEATURE = SPECTRAL_CHANGE
FEATURE = PERIODICITY
FEATURE = MOMENT_1ST
FEATURE = MOMENT_2ND
FEATURE = MOMENT_3RD
FEATURE = MOMENT_4TH

CLASS = 0,sil
CLASS = 1,ow
CLASS = 2,b
CLASS = 3,y
PHONEME = sil,ow,b,y

NODE = 0 = 2,tot,rel,per

NODE = 0.0 = 2,per
NODE = 0.1 = 2,tot,rel,mol

NODE = 0.0.0 = 0,C0
NODE = 0.0.1 = 0,C2

NODE = 0.1.0 = 0,C1
NODE = 0.1.1 = 0,C3
```

```
$ pre obeycb -v

Feature files being generated:
------------------------------
  ZERO_CROSSING (OBEY_FULL.zer)     TOTAL_ENERGY (OBEY_FULL.tot)
    PEAK_ENERGY (OBEY_FULL.pea)  RELATIVE_ENERGY (OBEY_FULL.rel)
SPECTRAL_CHANGE (OBEY_FULL.spe)      PERIODICITY (OBEY_FULL.per)
     MOMENT_1ST (OBEY_FULL.mo1)       MOMENT_2ND (OBEY_FULL.mo2)
     MOMENT_3RD (OBEY_FULL.mo3)       MOMENT_4TH (OBEY_FULL.mo4)


Phonemes found:
---------------
   sil = 1          ow = 16          y = 30          b = 8

Classes found:  0 = 1       Silence
                1 = 16      Strong frics
                2 = 8       Weak frics
                3 = 30      Vowels




$norm obeycb -v

 File            High     Low  Sub_shift     S. D.    Multiplier
-----------------------------------------------------------------
OBEY_FULL.zer      18       0        -9     4.729021  211.460273
OBEY_FULL.zer    1903   -1691       -30   999.078130    1.000923
OBEY_FULL.tot    2317       0     -1037   634.761430    1.575395
OBEY_FULL.tot    2016   -1540         0   999.571135    1.000429
OBEY_FULL.pea   22098       0    -10146  3899.829847    0.256421
OBEY_FULL.pea    3064   -1326         0   999.578866    1.000421
OBEY_FULL.rel    2513       0      -930   370.749830    2.697237
OBEY_FULL.rel    4269   -1993        -1   999.595300    1.000405
OBEY_FULL.spe   15223       0     -3480  2786.291377    0.358900
OBEY_FULL.spe    4214   -1205         0   999.611315    1.000389
OBEY_FULL.per     985       0      -774   181.868484    5.498479
OBEY_FULL.per    1160   -2001        -1   999.579457    1.000421
OBEY_FULL.mo1    2645       0     -1692   494.467427    2.022378
OBEY_FULL.mo1    1927   -1611         0   999.484767    1.000515
OBEY_FULL.mo2     272       0      -202    23.393861   42.746257
OBEY_FULL.mo2    2992   -2137        -3   999.587424    1.000413
OBEY_FULL.mo3     247       0      -121    79.848721   12.523682
OBEY_FULL.mo3    1577   -1327       -11   999.399874    1.000600
OBEY_FULL.mo4     567     -61      -149   198.684995    5.033093
OBEY_FULL.mo4    2103   -1056        -2   999.479301    1.000521
```

```
$ktrain obeycb -v

------------------------------------------------------------
Class to node association:  N3=C0 N4=C2 N5=C1 N6=C3
Method = Fixed classes  not resetting classes
------------------------------------------------------------

Results for node: 0
Performance:       54 out of 55  99%
Feature sets:      1
Feature vector:    ( tot rel per )

Node 1, SD = 1122  Children=(3-4)
  mean vector = ( -1403 1012 1066 )
 C0=1      C1=1      C2=8

Node 2, SD = 1388  Children=(5-6)
  mean vector = ( 275 -196 -206 )
 C1=15     C3=30


Node 1  Children  3-4
-------------------------
   sil     1    0   C0  (1)
     b     8    0   C2  (8)


Node 2  Children  5-6
-------------------------
    ow     1   15   C1  (16)
     y     0   30   C3  (30)


------------------------------------------------------------

Results for node: 1
Performance:       9 out of 9  100%
Feature sets:      1
Feature vector:    ( per )

Node 3, SD = 1000  Class=(0) Silence
  mean vector = ( 835 )
 C0=1

Node 4, SD = 56  Class=(2) Weak frics
  mean vector = ( 1095 )
 C2=8


Node 3  Class 0  Silence
-------------------------
   sil     1    0   C0  (1)

Node 4  Class 2  Weak frics
-------------------------
     b     0    8   C2  (8)
```

```
-------------------------------------------------------
Results for node: 2
Performance:      45 out of 46  98%
Feature sets:     1
Feature vector:   ( tot rel mol )

Node 5, SD = 783  Class=(1) Strong frics
  mean vector = ( 51 -80 -912 )
 C1=16     C3=1

Node 6, SD = 1121  Class=(3) Vowels
  mean vector = ( 395 -258 836 )
 C3=29


Node 5  Class 1  Strong frics
----------------------
    ow    16    0   C1  (16)

Node 6  Class 3  Vowels
----------------------
     y     1   29   C3  (30)

-------------------------------------------------------
-----------------  Trained Classes  ------------------

Class = 0 Silence
   pau   sil   pcl   tcl   kcl   qcl  dhcl   m-h   w-h   w-q

Class = 1 Strong frics
    ch    sh    jh    zh     s     z     t     d   t-h   t-b
   d-b    dh     v    ow

Class = 2 Weak frics
   p-b   b-b   g-b   ncl     f   r-h    th   k-h     b

Class = 3 Vowels
    uh    ao    aa    ey    ay    oy    aw     e     o    ih
    eh    ae    ux    oe    ax    ah  ih-n     r  ix-q  ey-q
  ae-q  aa-q    er   axr    el     l  aw-n    iy   l-h     y
  uw-q  eh-q     w    uw

Class = 4 Closures
  ih-q   bcl   dcl   gcl     p     g   mcl  ngcl     m     n
  ix-q    ng  ng-b   q-b    nx    em    en   eng    dx    lx
    rx   r-q    hh   voi  er-q   p-h  iy-q     h     q    ph
     k   b-h   d-h   g-h   n-h  ng-h   q-h   k-b   m-b   n-b
    bh    kh    gh    hv    ix axr-q
```

```
$class obeycb -v

      E0 sil        E1 ow        C1 ow        C1 ow        C1 ow
      C1 ow         C1 ow        C1 ow        C1 ow        C1 ow
      C1 ow         C1 ow        C1 ow        C1 ow        C1 ow
      C1 ow         E1 ow     C2.E1 b      C2.C1 b         C2 b
      C2 b          C2 b         C2 b         C2 b      C2.E1 b
   C3-E1 y          C3 y         C3 y         C3 y         C3 y
      C3 y          C3 y         C3 y         C3 y         C3 y
      C3 y          C3 y         C3 y         C3 y         C3 y
      C3 y          C3 y         C3 y         C3 y         C3 y
      C3 y          C3 y         C3 y         C3 y         C3 y
      C3 y          C3 y         C3 y         C3 y

Analysis by frame

          Sil     SF    WF   Vow                       Sil     SF    WF    Vow
      +-----------------------+                  +-----------------------+
  Sil | 100%     0%    0%    0% |  1      Sil  |  0%     0%    0%     0% |  0
   SF |   0%   100%    0%    0% | 16       SF  |  0%   100%    0%     0% | 14
   WF |   0%    37%   62%    0% |  8       WF  |  0%    17%   83%     0% |  6
  Vow |   0%     3%    0%   96% | 29      Vow  |  0%     0%    0%   100% | 27
      +-----------------------+                  +-----------------------+
       With edges    94%                        Without edges     98%

                              Analysis by frame

           ----------------------------------------------------------
           Sil      100%    100%    100%    100%    100%  out of   1
            SF      100%    100%    100%    100%      0%  out of   1
            WF      100%    100%    100%    100%      0%  out of   1
           Vow      100%    100%    100%    100%      0%  out of   1
           ----------------------------------------------------------
           All      100%    100%    100%    100%     25%  out of   4
```

APPENDIX D

D CLASSIFIER RESULTS

This appendix shows the detailed results of the six classifiers. The tables shown here are directly taken from the output produced by the classifiers as they were trained then tested on the full set of 98 utterances. Table D.1 shows two confusion matrices for each classifier. These matrices are showing the performance on a per frame basis. The first matrix includes every frame and the second includes only frames that are not within 10 msecs of a segment edge. The difference between the two gives some general information about where the errors are occurring. Table D.2 shows the details of the segment based error analysis. In this table each classifier reports five percentages for each class as well as for its overall performance. The five percentages represent the following:

A) Percentage of segments where at least some frames within the segment were labeled with the correct class.

B) Percentage of segments where one and only one contiguous string of frames within the segment were labeled with the correct class, regardless of how close to the actual edges this string came.

C) Percentage of segments all the frames were labeled correctly.

D) Percentage of segments all the frames were labeled correctly except those within 10 milliseconds of the edge of the segment.

E) Percentage of segments all the frames were labeled correctly except those within 20 milliseconds of the edge of the segment.

K-means - Single level tree

	Sil	SF	WF	Vow			Sil	SF	WF	Vow	
Sil	54%	2%	20%	24%	2532		63%	0%	15%	21%	1603
SF	8%	74%	11%	7%	6165		6%	84%	8%	3%	4972
WF	20%	14%	48%	18%	2576		19%	17%	54%	9%	1889
Vow	8%	1%	4%	87%	15329		7%	0%	2%	90%	13517
	With edges 77%						Without edges 84%				

K-means - Binary tree

Sil	71%	4%	5%	21%	3103		78%	2%	5%	16%	2100
SF	9%	75%	8%	9%	6165		6%	83%	8%	4%	4972
WF	31%	15%	40%	15%	1909		29%	17%	46%	8%	1518
Vow	10%	2%	1%	87%	15425		9%	1%	1%	90%	13611
	With edges 79%						Without edges 84%				

K-means - Skewed tree

Sil	65%	4%	9%	22%	3532		71%	3%	8%	18%	2261
SF	6%	83%	6%	6%	6309		3%	91%	5%	2%	5064
WF	19%	23%	46%	12%	2439		15%	27%	52%	6%	1849
Vow	13%	1%	2%	85%	14322		12%	0%	1%	86%	12300
	With edges 78%						Without edges 83%				

Maximum likelihood - Single level tree

Sil	88%	4%	5%	4%	2533		93%	1%	4%	2%	1603
SF	8%	74%	15%	3%	6166		4%	83%	12%	1%	4973
WF	32%	10%	55%	3%	2576		26%	11%	62%	1%	1889
Vow	19%	2%	15%	64%	15328		17%	1%	14%	68%	13516
	With edges 68%						Without edges 73%				

Maximum likelihood - Binary tree

Sil	65%	9%	17%	9%	3104		74%	5%	16%	5%	2100
SF	3%	86%	9%	2%	6166		2%	90%	8%	1%	4973
WF	12%	9%	72%	6%	1909		9%	9%	79%	3%	1518
Vow	13%	2%	3%	82%	15424		12%	1%	2%	85%	13610
	With edges 80%						Without edges 85%				

Maximum likelihood - Skewed tree

Sil	42%	11%	26%	21%	3532		52%	8%	25%	15%	2262
SF	2%	89%	8%	2%	6310		1%	94%	4%	1%	5065
WF	9%	46%	38%	7%	2439		7%	53%	38%	3%	1849
Vow	7%	2%	5%	86%	14322		6%	0%	4%	89%	12300
	With edges 76%						Without edges 82%				

Table D.1 - Confusion matrices showing frame by frame analysis results of all six classifiers.

K-means - Single level tree

Class	A	B	C	D	E		Segments
Sil	69%	66%	17%	37%	51%	out of	479
SF	91%	89%	9%	45%	73%	out of	597
WF	70%	62%	8%	22%	39%	out of	362
Vow	99%	85%	36%	66%	76%	out of	908
All	87%	79%	21%	48%	65%	out of	2346

K-means - Binary tree

Sil	83%	79%	32%	56%	66%	out of	517
SF	92%	89%	9%	35%	68%	out of	597
WF	73%	65%	1%	6%	19%	out of	196
Vow	97%	86%	34%	64%	74%	out of	912
All	91%	83%	24%	49%	66%	out of	2222

K-means - Skewed tree

Sil	83%	77%	31%	50%	63%	out of	653
SF	95%	93%	24%	66%	83%	out of	623
WF	69%	61%	6%	21%	33%	out of	301
Vow	98%	83%	45%	67%	74%	out of	1015
All	90%	81%	32%	57%	68%	out of	2592

Maximum likelihood - Single level tree

Sil	91%	88%	64%	82%	85%	out of	480
SF	92%	89%	11%	48%	70%	out of	597
WF	70%	62%	10%	25%	38%	out of	362
Vow	86%	68%	12%	34%	42%	out of	908
All	86%	76%	22%	46%	57%	out of	2347

Maximum likelihood - Binary tree

Sil	77%	72%	24%	49%	60%	out of	518
SF	95%	93%	48%	70%	80%	out of	597
WF	92%	86%	15%	44%	62%	out of	196
Vow	96%	82%	33%	58%	64%	out of	912
All	91%	83%	33%	58%	67%	out of	2223

Maximum likelihood - Skewed tree

Sil	59%	56%	8%	28%	39%	out of	653
SF	96%	95%	44%	73%	89%	out of	623
WF	77%	65%	12%	24%	31%	out of	301
Vow	96%	85%	40%	69%	74%	out of	1015
All	85%	78%	29%	54%	64%	out of	2592

Table D.2 - Segment by segment analysis results of all six classifiers.

APPENDIX E

E UTTERANCE TRAINING SET

This appendix lists the utterances used for training and testing the classifiers described in this paper. These were all taken from a larger set complied by Carnegie Mellon University. The file names shown here are those that were used by Carnegie Mellon. There is a total of 98 utterances spoken by 10 speakers, 5 female and 5 male.

```
/usr/tidb/fricdata/ADC/flt66/F3.1.adc  to  F3.10.adc
```

1 Zanzibar is a lovely island off the coast of Africa.
2 Voodoo influences the lives of superstitious peasants.
3 A Schick shaver is best for removing facial hair.
4 She'll choose whomever the association approves.
5 Susie wears a size five in socks.
6 Festus vouched for the authenticity of the check.
7 Thick haze enshrouded the hostile valley.
8 The czars only foible was his thirst for sweet wine.
9 Fred's search for salvation sent him to Zen philosophy.
10 The sergeant shoved the volunteers to the front.

```
/usr/tidb/fricdata/ADC/fwr66/F4.1.adc  to  F4.10.adc
/usr/tidb/fricdata/ADC/mjp66/F4.1.adc  to  F4.10.adc
/usr/tidb/fricdata/ADC/mdc66/F4.1.adc  to  F4.10.adc
(missing F4.3.adc)
```

1 Thoughtful viewers like those types of shows.
2 Shirley is the zaniest violinist in Ashland.
3 The shifty huckster fleeced his hapless victim.
4 The soot showered down the chimney of the haunted house.
5 Ancient civilizations are most visible through their artifacts.
6 Persuasion is the art of swaying the sure.
7 Schuster hoisted the heavy hose to his shoulder.
8 Sophisticated shysters vie for foolish souls.
9 Fewer than one thousand votes were needed to defeat the charismatic judge.
10 Joseph usually avoids the pistachio fudge.

```
/usr/tidb/fricdata/ADC/ftc66/F5.1.adc  to  F5.10.adc
/usr/tidb/fricdata/ADC/mmj66/F5.1.adc  to  F5.10.adc
```

1 A fuse shattered and the huge house was left in shadows.
2 The thrush constructs her nest from feathers and pieces of fluff.
3 His father sold the vase before they realized its value.
4 Cheech and Chong have unusual voices.
5 Treasure hunts make me vomit.
6 The sound of the zither gave Shirley a sensation of deja vu.
7 The freezer always thaws the food.
8 Should we rehearse the third scene again?
9 Heath is a shrub that thrives in the Scottish highlands.
10 Discussion of Zionism caused him to suffer from ulcers.

```
/usr/tidb/fricdata/ADC/fhe66/F6.1.adc  to  F6.10.adc
/usr/tidb/fricdata/ADC/miy66/F6.1.adc  to  F6.10.adc
```

1 The thief groped his flashlight out of nervousness.
2 Mother fixed the thatch roof before the thunder and hail struck.
3 The sound of the horses' hooves thundered through the evening silence.
4 Thistles and sunflowers are both in the daisy family.
5 Athena sprang from the head of Zeus.
6 Every first child gets everything he wants.
7 Blackbeard seized the English admiral's flag ship.
8 This soy sauce has no artificial ingredients or preservatives.
9 The foul ball sailed over the stadium roof.
10 Thanksgiving symbolizes the fall harvest feast here in the U.S.

```
/usr/tidb/fricdata/ADC/fmm66/F7.1.adc  to  F7.10.adc
/usr/tidb/fricdata/ADC/mes66/F7.1.adc  to  F7.9.adc
```

1 Athletic events jam the TV on Saturday afternoons.
2 The mazurkas were played with verve and enthusiasm.
3 Julie hooked a giant of a fish yesterday.
4 George and Faye are very fond of faraway vacations.
5 He injured his foot when he slid home.
6 The theft of the silver sousaphone had the sheriff baffled.
7 Herschel's visa for South Africa has yet to be approved.
8 Shoofly pie is a specialty of the Dutch of Pennsylvania.
9 The suicidal teenager slashed her wrist with a shard of glass.
10 The sheaves of wheat stood as sentinels in the snowy fields.

APPENDIX F

F CODE FOR FEATURE EXTRACTION

This appendix gives the code used for feature extraction in this work.

```
/**********************    feat.c   ***************************
*
*  FEAT.C
*
*
*  This file contains all the feature extraction routines for
*  the CLASS thesis, by Jim Delmege.
*
*  For detailed information on each feature see the header for
*  each feature extraction routine.
*
*
*  Common variables:
*/

unsigned int
  FFT_data[100][64],        /* FFT data (max FRAME 100 msec)    */
  FFT_avg[64],              /* Average FFT readings across frame*/
  FFT_hist[3][64]={0},      /* History of last three FFT_avgs   */
  PCM_per_frame=130;        /* Number of PCM samples in frame   */

int
  PCM_data[1300],           /* PCM data for frame (max 100 msec)*/
  PCM_shift[1300];          /* Shifted version of the PCM signal*/

short
  feat_value=0;             /* Feature value returned           */

double
  moment[5];                /* Value of the moments             */
```

```
/*************************************************************
*
*  This function will return the zero crossing count of a
*  given frame of raw PCM from a sample of speech.
*
*  A dead band is used which can be set in the options file of
*  the program.  A count is added on the positive the negative
*  edge of the signal, when it passes completely through the
*  dead band.
*/

extract_zero_crossing()
{
  unsigned int positive=TRUE,      /* Current state of signal */
               i;                  /* General index variable  */

  feat_value = 0;                  /* Initialize zero count   */
  for (i=0; i!=PCM_per_frame; ++i) { /* For each PCM sample   */
    if ( (positive == TRUE)  &&    /* If was POS and now NEG  */
        (PCM_data[i] < dead_band_low)) {
      positive = FALSE;            /* Set status to NEG       */
      feat_value = feat_value + 1; /* Add one to zero count   */
    }
    if ( (positive == FALSE) &&    /* If was NEG and now POS  */
        (PCM_data[i] > dead_band_high)) {
      positive = TRUE;             /* Set status to POS       */
    }
  }
}


/*************************************************************
*
*  This function will return the total energy of a
*  given frame of FFT's from a sample of speech.
*
*  The FFT_avg contains the average FFT's for each frequency
*  band, and thus total only needs add up the bands.  This
*  will return an average FFT total, rather than the true total.
*  Also, a band width can be specified so as to narrow the range
*  of bands viewed.
*/

extract_total_energy()
{
  unsigned long  total=0;
  int    j;

  for (j=total_low/FFT_BAND; j<total_high/FFT_BAND; ++j)
    total = total + FFT_avg[j];
  feat_value = total /
             ( (total_high/FFT_BAND)-(total_low/FFT_BAND) );
}


/*************************************************************
```

```
*
*  This function will return the peak energy relative
*  to the total energy of a
*  given frame of FFT's from a sample of speech.
*/

extract_peak_energy()
{
  unsigned long  sum[64],
                 large = 0,
                 total = 0;
  unsigned int   j;
  double         temp;


  for (j=0; j<64; j++) sum[j] = 0;  /* Clear out the sum array */

  for (j=0; j<64; j++) {            /* Get the total energy    */
    total  = total + FFT_avg[j];
    if (large < FFT_avg[j]) {
      large = FFT_avg[j];
    }
  }
  if ( large == 0 ) {               /* Avoid divide by zero    */
    feat_value = 0;
  } else {
    if ( large == 0 )
      temp = 0;
    else
      temp = (double) total / large; /* Get the relative number*/
    temp = temp * 1000;             /* Get 3 significant digits*/
    feat_value = temp;              /* Keep only non-fractional*/
  }
}
```

```
/**************************************************************
*
*  This function will return the relative energy of a
*  given frame of FFT's from a sample of speech.
*
*  Relative energy calculation will compare the energy in any
*  two ranges as specified in the options file.
*
*/

extract_rel_energy()
{
  unsigned long  low_total=0,
                 high_total=0;
  unsigned int   j;
  double         temp;

/* Low range energy */
  for (j=rel_low_low/FFT_BAND; j<rel_low_high/FFT_BAND; ++j)
    low_total = low_total + FFT_avg[j];

/* High range energy */
  for (j=rel_high_low/FFT_BAND; j<rel_high_high/FFT_BAND; ++j)
    high_total = high_total + FFT_avg[j];

  if ( low_total == 0 ) {                    /* Avoid divide by zero */
    feat_value = 0;
  } else {
    if ( low_total == 0 )
      temp = 0;
    else
      temp = (double) high_total /
                      low_total;             /* Get relative number  */
    temp = temp * 1000;                      /* 3 significant digits */
    feat_value = temp;                       /* Keep only nonfraction*/
  }
 }
```

```
/****************************************************************
*
*   This function will return the spectral change of the energy
*   in a given frame of FFT's from a sample of speech.
*
*
*   This is a summation of the differences between the current
*   frame and the previous three frames.
*
*/

extract_spectral_change()
{
  unsigned long    avg_hist[64],     /* Ave. of the last 3 frames */
                   diff = 0;         /* Total differences         */
  unsigned int     i;                /* Index variable            */


  for (i=spec_low/FFT_BAND; i<spec_high/FFT_BAND; ++i) {
    avg_hist[i] = FFT_hist[0][i];       /* Add up in the history */
    avg_hist[i] = avg_hist[i] + FFT_hist[1][i];
    avg_hist[i] = avg_hist[i] + FFT_hist[2][i];
    avg_hist[i] = avg_hist[i] / 3;
    if ( avg_hist[i] > FFT_avg[i] ) {        /* Find difference */
      diff = diff + ( avg_hist[i] - FFT_avg[i] );
    } else {
      diff = diff + ( FFT_avg[i] - avg_hist[i] );
    }
  }
  diff = (diff*10) / ((spec_high/FFT_BAND)-(spec_low/FFT_BAND));
  feat_value = diff;
}
```

```
/*************************************************************
*
*  This function will return the periodicity of the speech
*  signal.  This is a comparison of the PCM signal against
*  a shifted version of itself.
*/

extract_periodicity()
{
int       i;
double    r,
          rx,
          ry,
          sumx    = 0,
          sumy    = 0,
          sumxy   = 0,
          sumxsqr = 0,
          sumysqr = 0,
          num,
          d1,
          d2,
          denom;

  for (i=0; i!=PCM_per_frame; i++) {
    rx        = PCM_data[i];
    rx        = rx      / 1000;
    sumx      = sumx    + rx;
    sumxsqr   = sumxsqr + ( rx * rx );

    ry        = PCM_shift[i];
    ry        = ry      / 10;
    sumy      = sumy    + ry;
    sumysqr   = sumysqr + ( ry * ry );

    sumxy     = sumxy   + ( rx * ry );
  }
  num         = sumxy   - ( sumx * sumy / PCM_per_frame );
  d1          = sumxsqr - ( sumx * sumx / PCM_per_frame );
  d2          = sumysqr - ( sumy * sumy / PCM_per_frame );
  denom       = sqrt( d1 * d2 );
  r           = ( num / denom ) * 1000; /* Return 3 significant */
  feat_value = r;
}
```

```
/****************************************************************
*
*   Moments()
*   Where order = 1 - mean       2 - variance
*                 3 - skewness   4 - kurtosis
*   Note: If moments are being used then the first moment MUST
*         be calculated before attempting to calculate y of
*         the second and the second must be calculated before
*         attempting the third or fourth.
*/
extract_moments()
{
int
           j,i;              /* Index variables                    */
double
           p,                /* Normalized power spectrum          */
           f,                /* Frequency                          */
           m,                /* Current moment                     */
           d,dl,
           fl;               /* Frequency prime (temp storage)     */

 m = 0;
 moment[1] = 0;  moment[2] = 0;  moment[3] = 0;  moment[4] = 0;

 if ( FFT_total_energy != 0 ) {
  for (i=0; i!=64; i++) {
   m = m + ((double)i*((double)FFT_avg[i]/FFT_total_energy));
  }
 }

 moment[1] = (m * 100);

 if ( FFT_total_energy != 0 ) {
  for (i=0; i!=64; i++) {
   dl = i - m;
   d = dl * dl;
   moment[2] =+ ( d * ((double)FFT_avg[i] / FFT_total_energy) );
   d = d * dl;
   moment[3] =+ ( d * ((double)FFT_avg[i] / FFT_total_energy) );
   d = d * dl;
   moment[4] =+ ( d * ((double)FFT_avg[i] / FFT_total_energy) );
  }
 }
 if ( moment[2] == 0 ) {
  moment[3] = 0;  moment[4] = 0;
 } else {
  moment[3] = ( moment[3] / sqrt(moment[2] *
                moment[2] * moment[2]) ) * 100;
  moment[4] = ( ( moment[4] / (moment[2] *
                  moment[2] ) ) - 3 ) * 100;
```

APPENDIX G

G CODE FOR K-MEANS CLUSTERING

This appendix gives the code used for the K-means clustering in this work.

```
/***********************    kmeans.c   ************************
*
*  This file contains the main routines needed for the K-means
*  process.  It consists of the following:
*
*   distribute_data()          - Distributes all the data points
*                                around the current center points.
*   get_new_centers_and_sd()   - Calculates the new center points
*                                and the standard deviations for
*                                each cluster.
*   get_init_centers()         - Calculates initial center points.
*   ktrain()                   - This is the K-means routine as used
*                                for training coarse classes.
*
```

```
********************   distribute_data()   ******************
*
*  This routine will take all the data points associated with
*  the current node and change their association according to
*  how close they are to each of the cluster current centers.
*
*  For each data point associated with this node
*    For each child node off this node
*       calculate distance from data point to center of child
*       if closest so far
*         place data point on this node
*       end if
*     end for
*  end for
*
*/
distribute_data()
{
int      i,
         j,
         k;

double   best_dist,
         dist;

unsigned
long   subtotal,
       total;

 for (i=0; i!=data_count; ++i ) { /* for each data point          */
  if (node_assoc[i] == node ) {   /* if dealing with this child */
   for (j=node_b[node]; j!=node_b[node+1]; ++j ) { /* each child*/
    total = 0;
    for (k=0; k!=MAX_FEAT; ++k) { /* look at each feature         */
     if (curr_flag[k] == TRUE) {  /* Only add features we use    */
      subtotal = curr_centers[k][j] - data[k][i];
      total = total + ( subtotal * subtotal );
     }
    }
    dist = sqrt((double) total ); /* sqrt of sum of squares     */
    if ( j == node_b[node] )      /* If first time through loop*/
     best_dist = dist + 1;        /* Force next if to be true   */
    if ( dist < best_dist ) {     /* if closest so far          */
     best_dist = dist;            /* remember this as best      */
     node_assoc[i] = j;           /* assoc. the data with node */
    }
   }
  }
 }
}
```

```
/****************   get_new_centers_and_sd()   ***************
*
*   Once the data has been distributed (ie clustered) this
*   routine will determine the centers and deviations.  The
*   centers are determined by taking the average of each
*   dimension individually.  The standard deviation is the
*   average difference between the center point and all
*   the data points in the cluster.
*
*/
get_new_centers_and_sd()
{
  int    i, j, k, div;
  long   total, run_total, subtotal, n;
  double dist;

  for (i=node_b[node]; i!=node_b[node+1]; ++i) {
    div = 0;
    for (j=0; j!=data_count; ++j)
      if (node_assoc[j] == i) ++div;
    for (j=0; j!=MAX_FEAT; ++j) {
      total = 0;
      for (k=0; k!=data_count; ++k) {
        if (node_assoc[k] == i) {
          total = total + data[j][k];
        }
      }
      if ( div != 0 )  total = total / div;
      new_centers[j][i] = total;
    }

/*  This next part gets the standard deviations  */
    run_total = 0;
    n = 0;
    for (j=0; j<data_count; ++j ) {
      if ( node_assoc[j] == i ) {
        total = 0;
        for (k=0; k!=MAX_FEAT; ++k) {
          if (curr_flag[k] == TRUE) {
            subtotal = data[k][j] - new_centers[k][i];
            total = total + ( subtotal * subtotal );
          }
        }
        dist = sqrt((double) total );   /* sqrt of sum of squares */
        run_total = run_total + dist;
        ++n;
      }
    }
    if (n==0)
      curr_sd[i] = standard_deviation;
    else
      curr_sd[i] = run_total / n;
    if (curr_sd[i] == 0) curr_sd[i] = standard_deviation;
  }
}
```

```
/*******************   get_init_centers()   *******************
*
*  The initial centers are determined by clustering the phonetic
*  samples by their initial coarse class and then determining
*  the centers of these clusters.  The clustering uses the coarse
*  classes as specified in the options file.  This may not be
*  their final coarse class but it is expected only to be a good
*  place from which to start the process.
*
*/
get_init_centers()
{
  int  i;

  cluster_by_class();
  get_new_centers_and_sd();
  for (i=0; i!=data_count; ++i)        /* For each data point */
    node_assoc[i] = back_assoc[i];     /* Restore node assoc. */
```

```
/*********************    kmeans()   ***************************
*
*  K-Means training routine
*
*
* Key variables:
*    node                  - current node being worked on
*                            (only those data points in the
*                            current node will be used).
*    node_k[node]          - number of branches for this node.
*    node_b[node]          - beginning node of the children of 'node'.
*    node_assoc[i]         - node to which vector 'i' is associated.
*    back_assoc[i]         - node to which vector 'i' was associated
*                            at the start of this procedure.
*    curr_centers[i][j]  - current center point where 'i' is the
*                            feature or vector dimension and 'j' is
*                            the node this vector will represent.
*    new_center[i][j]      - newly calculated center.
*    curr_flag[i]          - TRUE/FALSE flag to indicate if a
*                            feature is being used or not.
*
*
* Process:
*        New centers  =  initial centers
*        Repeat
*           Current centers  =  New centers
*           Distribute all the data around the center points
*           New centers  =  centers of the new clusters
*        Until New centers  ==  Current centers
*
*
* Results:
*        curr_centers - current cluster centers for this pass
*
*
* Note:
*        For more details see section called "The clustering
*        algorithm, K-means", in my thesis. "CLASS - A Coarse
*        Phonetic Classifier", 1988.  Also, the text "Pattern
*        Recognition Principles", by J. Tou and R. Gonzalez,
*        1974 contains this and other clustering algorithms.
*/
```

```
ktrain()
{
  int done,
      i,
      j;


 get_init_centers();
 do {                          /* loop until centers don't change */
                               /* For each child node             */
  for ( j=node_b[node]; j!=node_b[node+1]; ++j) {
   for ( i=0; i!=MAX_FEAT; ++i) {      /* For each feature        */
    curr_centers[i][j] = new_centers[i][j]; /* make them curr     */
   }
  }

  for (i=0; i!=MAX_DATA; ++i)      /* Restore node associations */
   node_assoc[i] = back_assoc[i];

  distribute_data();           /* Distribute around curr centers */
  get_new_centers_and_sd();    /* Calculate new centers and dev. */

  done = TRUE;                         /* Assume we are done      */
  for ( j=node_b[node]; j!=node_b[node+1]; ++j) {
   for ( i=0; i!=MAX_FEAT; ++i) {      /* Look at each feature    */
    if (curr_flag[i] == TRUE) {
                                       /* If a center changed     */
     if (curr_centers[i][j]!=new_centers[i][j]) {
      done = FALSE;                    /* then we are not done    */
     }
    }
   }
  }
  k_means_count++;              /* Counter for the fun of it     */
 } while ( done==FALSE );       /* While the centers don't match */
}
```