Graphical Analysis of Hidden Markov Model Experiments

by

DeWitt C. Seward IV

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author ....
Department of Electrical Engineering and Computer Science
May 6, 1994

Certified by ....
Victor W. Zue
Principal Research Scientist, Associate Director, Laboratory of Computer Science
Thesis Supervisor

Certified by ....
Marc A. Zissman
Staff Member, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by ....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Graphical Analysis of Hidden Markov Model Experiments

by

DeWitt C. Seward IV

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 1994, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Engineering

Abstract

Hidden Markov models are powerful tools for acoustic modeling in speech recognition systems. However, detailed analysis of their performance in specific experiments can be difficult. Two tools were developed and implemented for the purpose of analyzing hidden Markov model experiments: an interactive Viterbi backtrace viewer, and a multi-dimensional scaling display. These tools were built using the HMM Toolkit (HTK). Use of the Viterbi backtrace tool provided insight that eventually led to improved recognition performance.

Thesis Supervisor: Victor W. Zue
Title: Principal Research Scientist
Associate Director, Laboratory of Computer Science

Thesis Supervisor: Marc A. Zissman
Title: Staff Member, MIT Lincoln Laboratory

Acknowledgments

I would like to thank my company thesis advisor, Dr. Marc A. Zissman, for all of his direction and support. He insured that my entire internship at MIT Lincoln Laboratory was a wonderful experience. Without his help and encouragement, none of this would have been possible.

I would also like to thank my campus supervisor, Dr. Victor W. Zue, for his guidance. His experience and knowledge were an invaluable asset in this endeavor.

Many thanks go to Group 24 at MIT Lincoln Laboratory, especially those with whom I discussed many ideas at length and solved many problems associated with this thesis: Jerry O'Leary, Dr. Beth A. Carlson, Dr. Richard P. Lippman, Dr. Douglas A. Reynolds, Terry P. Gleason, Charles R. Jankowski, Eric I. Chang, and Linda Sue Sohn.

Finally, I would also like to thank those who have helped me throughout my MIT experience. I would like to thank Judy, my friends, and my family, all of whom helped to make my MIT dreams come true. Special thanks go to my parents, who have always provided unconditional love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hidden Markov Models (HMM) are a powerful tool for performing pattern recognition [10]. They are especially useful for speech recognition and word spotting. But, that power comes at a price. HMM systems can be quite complex, making their creation and analysis both time consuming and difficult. While software packages that ease the creation and execution of HMM experiments exist, the facilities for analyzing the results of such experiments in order to gain insight into successes and failures are not as well developed. Described in this thesis are some data analysis tools, integrated into an existing HMM software package, which assist in the analysis of HMM experiments.

1.1 The Problem of Examining Experiments

At Lincoln Laboratory, the Hidden Markov Model ToolKit (HTK) is often used to create and run HMM experiments [16]. HTK is a set of libraries and programs designed to facilitate the creation, running, and evaluation of HMM experiments. These experiments are evaluated mainly on the basis of recognition performance, and intermediate data analysis is typically not performed. Because of the complexity of HMM systems, there is often no easy way to analyze this information to determine more details about where or why a model was effective or ineffective. Although HTK can output vast amounts of diagnostic information and its models are defined in great detail, there is no method for easily interpreting this information. Additionally, HTK does not always keep track of all of the important intermediate data during the recognition experiments. So, while HTK eases the creation and running process for HMM experiments, there exist no set of tools which facilitate data analysis.

1.2 Exploratory Data Analysis

One of the goals of data analysis is to open up data so that it can tell its own story [15]. Hopefully, after creating unbiased pictures from diagnostic data, a researcher can gain a better understanding for what the system is actually doing. As this understanding improves, one should be able to formulate more easily ideas about current system limitations and areas for possible improvement. Therefore, one important goal for creating data analysis tools is to realize techniques which are general and unbiased enough to permit widely different interpretations of the data.

Eventually, the goal of any analysis is to improve the subject of the analysis, and a prototypical data analysis process is as follows [8]. First, the system is run. Next, intermediate data is studied using data analysis techniques. Eventually, improvements are suggested and implemented. Earlier design decisions or assumptions are replaced with those made from a better understanding of the problem and the system's actual implementation. The final step in this process should be to evaluate any new design decisions to determine if they produce the desired results. So, the data analysis techniques should be able to suggest improvements.

With these goals in mind, a few criteria for analysis techniques may be formulated. First, a system should not be biased toward an answer beforehand. It should not seek to prove or disprove too specific a theory about the system [15]. Rather, the system must strive for unbiased analysis. Second, the techniques should display useful information. If the data displayed is too general for meaningful analysis, the technique fails. Third, any implemented systems should be easy to use. Nothing is gained when a data analysis technique has been developed but nobody can use it [15]. Finally, to be useful at Lincoln Laboratory, the technique had to be integrated into the HTK system. As it turns out, some of the methods were implemented such that they should be easily portable to other HMM systems with a relatively small amount of effort, although no such effort has been made.

1.3 Overview of Analysis Tools

This thesis created and used two data analysis techniques. One system produced the Viterbi backtrace for the putative hits of a recognition experiment (see Chapter 3). This backtrace included information such as word model design, truth, state occupancy, as well as the per-state average of the normalized log likelihood scores. The other technique involved using some existing software as well as creating some new software to produce two dimensional pictures of multidimensional data (see Chapter 4). Additionally, a few examples of data analysis experiments were run and evaluated.

Chapter 2

Background

2.1 Introduction to Hidden Markov Models

Hidden Markov Models (HMMs) provide useful models for speech production. The different states in a HMM allow the modeling of speech as sounds progress. However, the complexity of the models is as much a burden as an asset when it comes to implementing the training and testing of a speech recognition system. To combat this problem, many researchers at Lincoln Laboratory use the Hidden Markov Model Toolkit (HTK) for running HMM experiments. This set of tools allows easier creation of HMM experiments so that more time can be spent on developing the general architecture of a recognition, rather than the details of the HMM training and recognition algorithms.

2.1.1 General Hidden Markov Model Introduction

Markov systems model future events as dependent on past events. At any time the model can be in one of several discrete states. According to a set of probabilistic rules, the system may, at certain discrete instants of time, undergo changes of state [2]. An example of a process that could be Markov modeled is a family's fast food restaurant choice: McDonald's or Burger King. As shown in Figure 2-1, the states of this model are the `B` state, or Burger King state, and the `M` state, or McDonald's

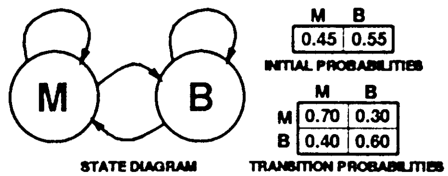

Figure 2-1: Example of a Simple Markov Model

state. At first, the family has never eaten at either McDonald's or Burger King, so they only have advertising and word of mouth to determine their initial choice. This is modeled with the initial probability vector. Based on factors, such as habit, additional advertising, and actual experience with the restaurant, the family's choice may change from night to night. The transition probability matrix models the combination of all of these factors, where the probability $P(M@t|B@t-1)$ is shown in row `B`, column `M` by convention. The process behind the family's choice can be modeled as a simple Markov model, and the actual choice would be the output of that model.

Markov models can model spoken words as well. Consider the possible model for the word "book" shown in Figure 2-2. A sample Markov model might consist of three states: /b/, /U/, and /k/. Additionally, the model would require a vector

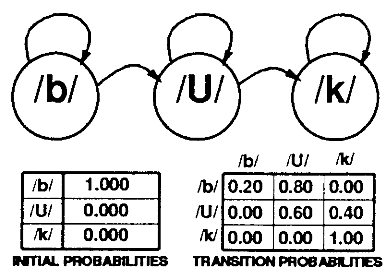

Figure 2-2: Example of a Simple Word Markov Model

of initial state probabilities and a matrix of state transition probabilities. Because the model for book begins with the /b/ sound and progresses left-to-right, the initial probabilities and transition probabilities are constrained. Just as the human mouth produces these sounds in succession, the model would transition through these three sounds one right after the other.

Both of these Markov models have produced output from which the state can be uniquely determined. For example, if we were eating at McDonald's, we are in state `M`.

A hidden Markov model adds the ability of a state to produce output probabilistically. Perhaps being in state M means we have a 85% chance of eating in McDonalds and a 15% chance of eating at Burger King. For this case, the state sequence can not be uniquely determined from the output in this case, thus it is referred to as "hidden". A hidden Markov model (HMM) produces observations according to a state-dependent probability density function [9]. Given this form, HMM users must address three important problems [10]:

1. Given the observations, $O = O_1, O_2, ..., O_T$, and the model, M, how do we compute $Pr(O|M)$?

2. Given the observations, $O = O_1, O_2, ..., O_T$, how do we choose a state sequence $I = i_1, i_2, ..., i_T$ which is optimal in some meaningful sense?

3. Given training observations, $O = O_1, O_2, ..., O_T$ that we assume came from a Markov process, how do we calculate the optimal model parameters?

The solution to these three problems is described in the following paragraphs.

Consider modeling an isolated word using an HMM, such as is illustrated in Figure 2-3. As before, there is a vector of initial state probabilities and a matrix of state

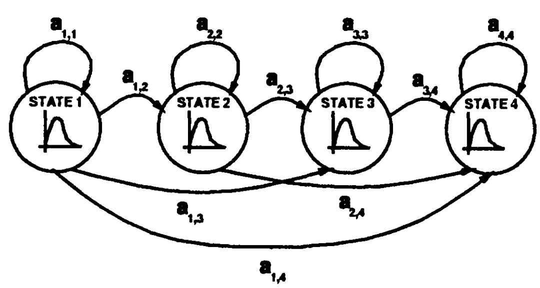

Figure 2-3: Example of a Left to Right Hidden Markov Model with Four States

transition probabilities. However, now the output is a probabilistically determined observation vector. The observation vectors might be spectral envelope or low order cepstral vectors and are generated by the probability density function of the model's current state. Additionally, for speech models, a left to right HMM is often used. This means disallowing backward transitions. For example, the probability of a transition from state 3 to state 2 would be set to zero. Also, the only initial state allowed is state 1.

In this way, a model for the word "book" can be constructed by specifying an HMM architecture with enough states to ensure that the different sounds may be sufficiently modeled, and training it on actual observation sequences from several spoken instances of "book" (see Figure 2-4). As before, each state represents the

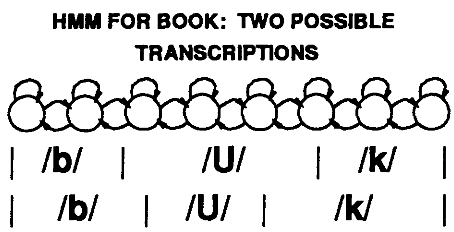

Figure 2-4: Hidden Markov Model of "book"

specific sounds spoken for the word "book". The difference is that now the model for the sounds is probabilistic. Another important feature of this model is that we have not specified where the different sounds began and ended. Figure 2-4 may have one of the two configurations shown, or it may have some completely unknown way of splitting the sounds. Those decisions are made by the system as it applies the HMM training algorithm.

With several models constructed for different words, the problem of identifying specific isolated words reduces to calculating the likelihood of a specific model given a set of speech observations $P(M|O)$. In practice, calculating $P(M|O)$ directly is an intractable problem, as it involves creating a probability model for every possible observation. Instead, $P(O|M)$ is calculated and used to produce $P(M|O)$ by applying Bayes Rule, seen in Equation 2.1,

$$P(M|O) = P(O|M) * \frac{P(M)}{P(O)} \tag{2.1}$$

Here, there are two extra terms, $P(M)$ and $P(O)$. $P(M)$ is the *a priori* probability of occurrence of this particular model, and can be estimated from the training data. $P(O)$ is a normalizing constant and is the same for all models. Since all of the $P(M|O)$

calculations are compared to each other for the final decision, this $P(O)$ term can be thrown out, as it affects each model probability the same. So, for calculating $P(M|O)$, we are left with Equation 2.2.

$$P(M|O) = P(O|M) * P(M) \tag{2.2}$$

The only term left to calculate is $P(O|M)$. This is done by recursively calculating the likelihood that the model is in state j at time t, as in Equation 2.3.

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{i,j} b_j(o_t) \tag{2.3}$$

Here, $\alpha_{t-1}(i)$ is the probability that the model was in state i at time $t-1$, $a_{i,j}$ is the matrix of transition probabilities from state i to state j, and $b_j(o_t)$ is the probability that o_t was produced by state j. In this way, the forward probabilities of the model, $\alpha_t(i)$, can be computed for the set of observations. Similarly, the backward probabilities of the model, $\beta_t(i)$ starting with the last observation can be computed for the same set of observations by recursively calculating $\beta_t(i)$'s. At time t, the probability that the set of observations was produced by the model is the sum over all states of the product of the forward and backward probabilities, seen in Equation 2.4 [9].

$$p(O|M) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i) \tag{2.4}$$

After the probability score is calculated for each model, the system can determine which model was most likely to have produced the observations.

Subword Modeling and Phones in Context

Sometimes, HMMs are used to model specific phonemes instead of complete words. It is these phoneme models that are concatenated to form the models of words, as shown in Figure 2-5. A word is recognized when its phoneme models lie, in proper order, on the most likely path through the network. Relating this to our "book" example, the HMMs of the system would model the /b/, /U/, and /k/ sounds instead of the

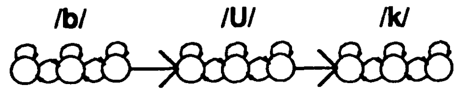

Figure 2-5: Example of "book" Modeled With Subword Models

word "book". A recognizer network including "book" would be built by connecting phoneme models for /b/, /U/, and /k/.

One problem with straight phone-based models is that phones are different depending on what sounds preceded or followed them. Because of this, the HMMs actually used to model important words in recognition systems are often phones in context. For our "book" example, the actual HMMs would not be `b`, U, and `k`. They would be `#-b+U`, `b-U+k`, `U-k+#`, where the `#` represents a word boundary. Here, `#-b+U` refers to the phone /b/ as it is said when preceded by silence and followed by the /U/ sound. Phone models that are specific to both the left and right contexts are called triphones.

2.1.2 Hidden Markov Model Toolkit: An Overview

The Hidden Markov Model Toolkit (HTK), created by Steve Young of Cambridge University, is a set of software tools designed for creating, training, and testing an HMM speech recognition system [16]. In addition to the software tools, it includes standards for the definition of, among other things, an HMM and an HMM network.

Creation of an HMM system begins with the definition of the HMM models themselves. For our system and task, described in Section 2.2.1, the word "card" was a particularly interesting word, because it occurred so frequently and because the system often confuses other words for "card". In order to model "card", four triphone models are necessary. These models are `#-k+aa`, `k-aa+r`, `aa-r+d`, and `r-d+#`, where `#` denotes a word boundary. Somewhere in the network definition, there is a representation of "card" which specifies these four triphones in succession.

Once all of the HMM models have been defined, the HTK tools can train the system. There are several tools for this purpose [16]. As this thesis did not focus on the training, it is sufficient at this time to state that they do exist. They will produce a trained set of hidden Markov models for use with the HTK recognizer.

After a new HMM system has been trained, it is time to use it in a recognition task. The main program used to recognize speech is `HVite`. `HVite` was the central focus of a major data analysis effort in this thesis (see Chapter 3). It implements a Viterbi decoder, which outputs the most likely words based on the best path through the network for all speech observations. Additionally, a specific peak picking mode was added at Lincoln Laboratory by making some additions to the original `HVite` program. This peak picking system was the subject of study in this thesis, and is described in Section 2.2.2.

2.2 Word Spotting

2.2.1 Word Spotting Using Subword Modeling and HTK

Word spotting is one class of speech recognition problems where the goal is partial transcription: the labeling of the occurrences of a few specific words in the speech waveform [13]. The specific task addressed is the "credit card" task based on the Switchboard telephone speech corpus [4]. In this task, the system looks for credit card type words in a conversation about credit cards. In Figure 2-6, we can see an

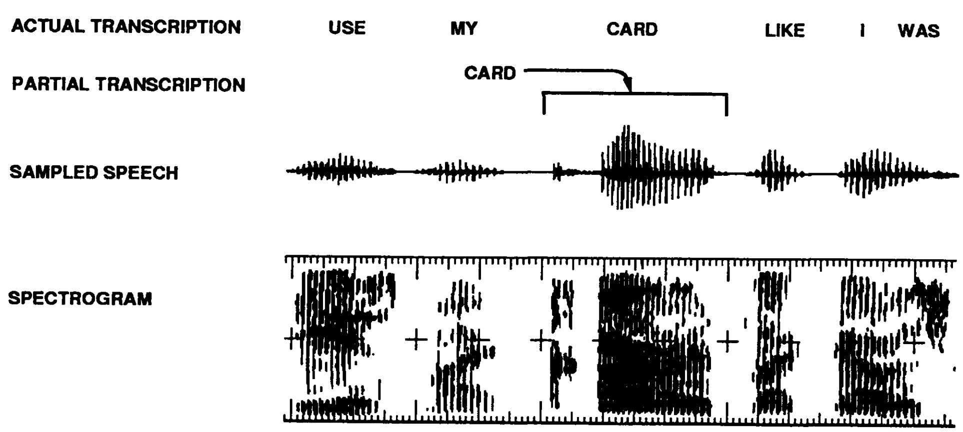

Figure 2-6: Example of Word Spotting for "card"

example of this. The system does not need to transcribe the entire utterance. It merely needs to find where specific words were spoken. In this case it could find the word "card". Since only a partial transcription of the speech is required, full Viterbi decoding is not necessary. Instead, only periods where specific words are likely need to be output. From this output, the system constructs a partial transcription of the speech message.

2.2.2 Peak Picking

Peak picking is one technique used at Lincoln Laboratory for word spotting. For each frame of speech, the system calculates the probability of a key word ending in that current frame. When this probability is plotted vs. time, the probability usually peaks when the word actually occurred. So, spotting a particular word reduces to the problem of finding periods where the probability of the word peaks above a specific threshold. These peaks are then picked (thus the name) as the locations of the end of the words.

One issue in peak picking is the problem of normalizing the scores of the HMM output. Since the decisions are no longer based on the best path through the network from the start of the speech segment to the end, peak scores need to be compared to a threshold value to determine if they should be labeled as a putative hit (see Figure 2-7). Because different segments of speech score differently depending on several

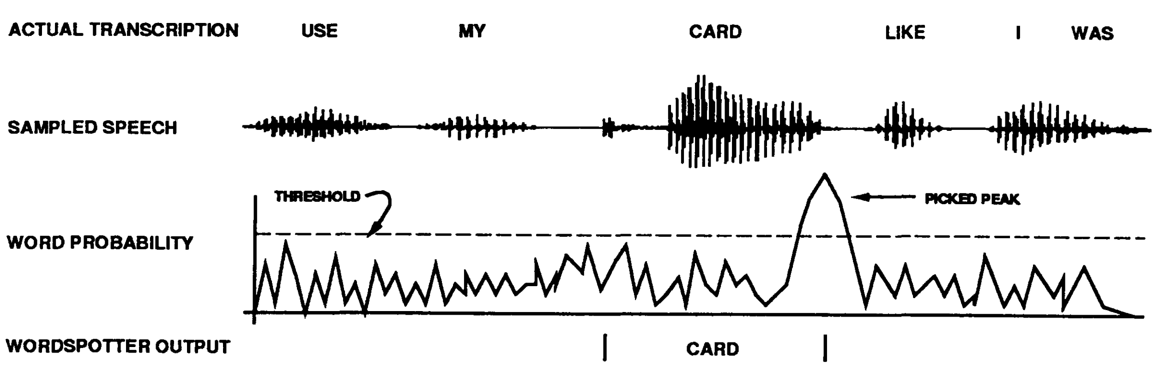

Figure 2-7: Example of Peak Picking for "card"

factors, such as speaker and communication channel, normalizing the score becomes quite important. This system implemented a technique which normalizes the score of model frames, $b_j(o_t)$ from Section 2.1.1, by subtracting a function of all of the state probabilities as follows:

$$b_j{}'(o_t) = b_j(o_t) - f(b_1(ot), b_2(ot), ..., b_N(ot)) \tag{2.5}$$

Several different forms of the function $f()$ are currently being explored. Normalizing relative to the probabilities of all of the states given the current observation should allow the score of the putative hit to reflect only the word being spoken and not other factors, such as specific speaker and channel.

After these scores are calculated, the figure of merit (FOM) for the word spotter is calculated to measure overall system performance. The figure of merit is based on how many true instances of the word were spotted on average when the threshold level is set to a specific false alarm rate, between 0 and 10 false alarms per key word per hour.

$$FOM = \int_0^{10} P_d(FA)dFA \tag{2.6}$$

Here, FOM is the figure of merit, FA is the threshold of the recognizer, as described by the rate of false alarms, and $P_d(FA)$ is the probability of detection based on the current false alarm rate.

2.2.3 Choice of the Word "card"

As has been mentioned in the preceding discussion, most of the data analysis work has been done for a particular word in the HTK word spotting system. One specific word was chosen because in the scope of this master's thesis, it was more important to develop the data analysis techniques than to improve word spotter performance, although the two issues are closely related as is stated in Section 1.2.

There were two main reasons for choosing "card" over the other 19 words in the credit card task. First, it is a frequently occurring word. Exploratory data analysis only makes sense where sufficient examples, both good and bad, exist. Otherwise, effects may be far too specific to a particular instance, and may have no general meaning. Several instances are necessary to prevent overgeneralizing based on a single token. Thus, a frequently occurring word is necessary for meaningful analysis.

Second, "card" is a poorly scoring word. Of the frequently occurring words, "card" is the one on which our HTK word spotter does worst. This, in theory, makes the data analysis easier. It is much easier to fix something clearly broken than something which works reasonably well. The bad performance of "card" suggests that it, more than other words, would be most interesting from a data analysis point of view. So, as "card" was the lowest scoring of the frequently spotted words, it was the choice for closer analysis.

Chapter 3

Viterbi Backtrace

3.1 Introduction

The first technique developed is a method for easily viewing the Viterbi backtrace of the experiment. The Viterbi backtrace is defined as the most probable path through the HMM trellis given the data observations. By looking at this backtrace, it is hoped that insights about where the system works and where it breaks down may be formulated.

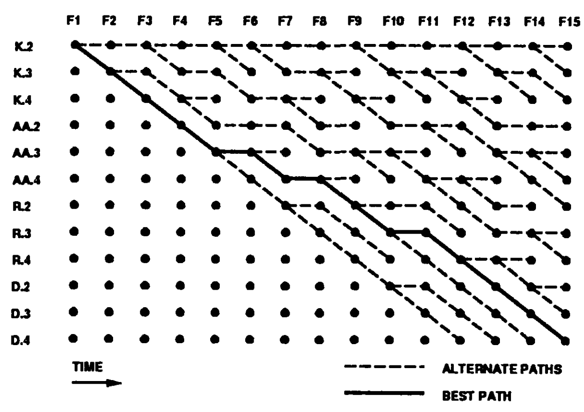

Figure 3-1: Example of Backtrace Decoding for "card"

In Figure 3-1, we see an example of a best path calculation as it progresses. Here, there are 15 frames of data which are being run against the model developed for "card". As can be seen, the system calculates several paths. Paths are pruned away either when they no longer fall along the best path to any node in the next frame of the decoding, or when their likelihood falls too far below the maximum of the likelihoods of the active paths. The backtrace tool seeks to display the overall best path in a meaningful way.

3.2 Implementation

The implementation of the backtrace tool proceeded in two phases. In the first phase, backtrace data was gathered. In the second phase, the backtrace viewing program was implemented. For data gathering, additions were made to the existing HTK Viterbi decoder. In addition, software was developed to create files containing other information which could provide context for the backtrace pictures. The backtrace viewer itself was written in C using the Motif library for the Graphical User Interface (GUI). This GUI eased the viewing of specific traces, and was therefore an integral part of the technique.

3.2.1 Data Gathering

In order for a complete backtrace to be drawn, several pieces of information are needed. First, the specific per state location and score along the best path through the system need to be output at the end of a trace gathering run. Additional pieces of important, but optional, information are listed here:

1. truth; list of what was really said during this time,
2. type; reference of whether a speech token was a hit, or a false alarm for ease of interface,
3. filename; name of specific conversation from which this piece of speech came from, necessary for handling multiple conversations at one time.

This additional information is needed to provide the pictures with some context.

Viterbi Backtrace

Drawing the backtrace required collecting detailed information from HTK. However, for efficiency reasons, the standard HTK decoder did not keep track of the full Viterbi backtrace. It only kept track of enough information to reproduce where specific models or words occurred along the best path. It did not store the more detailed information of specific state occupancies and specific frame scores in the backtrace path. A new data structure was therefore developed and added to the decoder to keep track of this information.

For our HTK systems, there were two methods of calculating when words occurred. In the first, Viterbi decoding, the best path through all data is calculated, and the system marks the key words wherever they occur in the best path. So, in reference to Figure 3-1, the information along the solid line would be output to a file, for later use by the backtrace viewing program. In peak picking, the second method, several different word models are being calculated at the same time. Whenever the score of the best path through these word models peaks above a certain threshold, the system produces a word mark. For this case, partial backtraces are produced as peaks are picked. In reference to Figure 3-1, this could either be information from the best path along solid lines, or from temporary paths along the dashed lines.

Since HTK's decoder program does not keep track of the backtrace down to individual frames of data, a specific backtrace data collection method needed to be developed. The solution was to create a large linked tree which resembles the backtrace picture in Figure 3-1. A picture of the actual structure is in Figure 3-2. To print

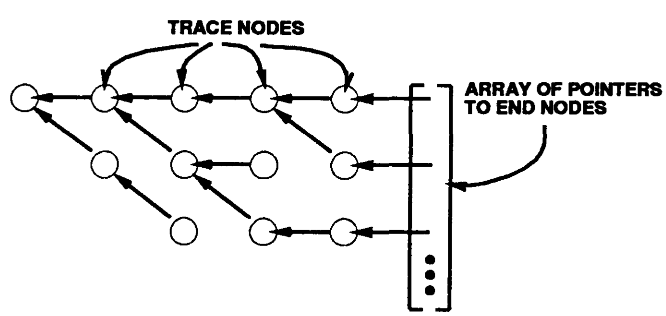

Figure 3-2: Picture of Trace Tree

out the backtrace information required, each node in the tree needed to keep track of the current frame, the model information (i.e. the model's name state for this frame of data), and the score that the current frame received given the system was in this particular state of this particular frame (i.e. the $b_j(o_t)$'s from Equation 2.5). In order to make this a tree, each node needed a pointer to the node of the previous time from which the best path to the current node in the current time came. The position of the final nodes is kept in an array, where each element in the array corresponds to each state in the HMM network.

At each frame, HTK would already calculate all the information needed to the backtrace. The new code only needed to create nodes for each new frame of data, and copy the proper information into the newly created nodes. Thus, the task was one of bookkeeping and not of calculation.

Eventually, as experiments became larger, a garbage collection scheme was developed. This was necessary as some experiments had upwards of 50,000 frames of data. Without garbage collection, with the 400 nodes in the system being used, and the 50 byte size of a node, the program would have used on the order of 10^9 bytes of memory. Thus, a field in each node which kept track of each successor was added. As successors to a node were added, the number of successors field would be incremented, and as successors were freed, the field would be decremented. When a branch of the tree had no successors, it was pruned and its predecessor was checked to see if it could be pruned as well. For example, if we refer to the example of Figure 3-1, after

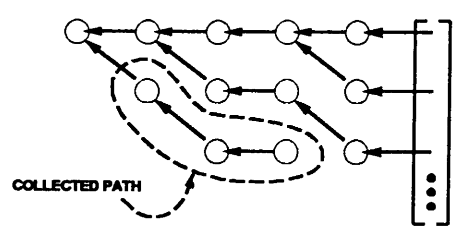

Figure 3-3: Garbage Collection in the Backtrace Output Code

frame 6 had been calculated, the branch ending in frame 5 for K.4 would be pruned, and the memory would be freed by the system. A picture of this is in Figure 3-3. By using this method, only important branches of the tree would be tracked. This resulted in reducing memory of the tree to below memory requirements of the rest of the program.

There were two methods for outputting this information, one for the peak picking decoder and one for the Viterbi decoder. The peak picking decoder would output trace information for each individual word as they were picked. These are written to a file with the `.pptrace` extension. The Viterbi decoder would have to wait until the entire best path had been calculated. Only at this time could decisions about the words be made, and then the backtrace could be output as well. This full back trace information is written to a file with a `.trace` extension.

Other Backtrace Information

While the backtrace contains much information, other sources were needed to completely classify examples of the backtrace, giving the backtrace pictures a context in which they can be interpreted. Methods for obtaining these additional pieces of information are detailed here.

The first useful piece of external information is the truth information. Truth refers to what was actually spoken by the person, in contrast to hypothesis, what the recognition system thinks was said. Files containing truth markings in an ASCII format were translated into a standard HTK format, so that the system could use the same code for reading in the truth markings as well as the HTK generated hypothesized markings. These translated files all contain the `.label` extension.

The second useful piece of information is the type of putative hit (i.e. was this token a true hit or a false alarm). Calculating putative hit type is a complex problem, as several different word variations may or may not be acceptable truth labels. For example, in the Lincoln system, "cards" or "mastercard" are acceptable variations of "card" and both signify a true hit when they coincide with the "card" hypothesis, but "credit card" is its own special word and, therefore, signifies a false alarm if it overlaps with a "card" hypothesis. As software for calculating the type of putative hit already existed, it was adapted for use in this thesis to produce files which the `vtrace` viewer could use to label putative hits. A program, called `rocplot`, which, among other things, calculates the figure of merit, and the type of each putative hit, had already been developed. It outputs the hit type for the peak picking case of the HMM decoding. A program included in the HTK package, called `HResults`, also produces output of this nature. The `vtrace` viewer used its output as well. The `rocplot` output was written to files with the `.sorted` extension, and the `HResults` file output was written to files with the `.results` extension, so the different types of putative hit files would not be confused.

3.2.2 Vtrace Viewer

Once all of the necessary information had been calculated and written to files on the system, the `vtrace` viewer could draw its analysis pictures based on these files. This program was written in C using the Motif library for the interface. It was written in two phases. The first phase developed the original tool which the author wrote. This provided an initial GUI interface, as well as code to read in, sort, select, and view the individual trace segments. The second ongoing phase involves the continuing effort to improve and expand upon the original definition of the backtrace viewer. This has been taken over by Linda Sue Sohn, a staff member of Group 24 at Lincoln Laboratory.

Operation Overview

The `vtrace` viewer program's general operation is described here. First, the user selects which trace file to view. When the selection has been made, `vtrace` loads all or some of the files specified in Section 3.2.1, depending on which are available. Next, it creates and presents to the user an ordered list of all putative hits. Then, the user clicks on the putative hit of interest, and `vtrace` displays it on the screen. The actual interface is detailed below.

It is important to note that not all of the files listed above are necessary for running `vtrace` viewer. In fact, the `vtrace` viewer only needs a `.trace` file to function. All other files are optional. This decision was made because truth or putative hit type are not always needed to gain insight from a trace, especially for some of the projects in the group outside the thesis. The only file needed to produce a trace is a `.trace` or `.pptrace` file.

Additionally, the `vtrace` viewer can order the list of putative hits based on several different parameters. The user has complete control over the sorting hierarchy (i.e., which parameters have which priority) through an additional window interface. These parameters are the time (beginning frame number), score of the putative hit, hit type, and hit label. A user could group all hits together to contrast them to all false alarms more easily. Alternatively, hits and false alarms of a specific word could be compared to determine why the false alarms occurred.

Implementation

As mentioned before, the program was written in C using the Motif library for the GUI. It was designed, written, and compiled using a Sun SPARCstation running X Windows. Most of the original Motif code was developed with the aid of the Builder Xcessory tool from Integrated Computer Solutions Inc.

Most of the work went into designing code to load all of the required files and to draw selected putative hits. The first half of this task involved loading the files, after determining which of the possible information files were available, creating the list of the putative hits, and sorting the hits list. Drawing the putative hit involved loading that segment of the trace file, calculating the position of the picture objects, and actually drawing the picture objects to the screen.

Loading Files

Loading files begins with a standard Motif filebox, seen in Figure 3-4. The routines for loading the files use the standard C library for input and output. All of the files have standard ASCII formats and are not too difficult to parse. Most of the files are too big, with the exception of the trace files, so no special algorithms are necessary. Because of the size of the trace files (upwards of 50,000 lines), an efficient loading method was developed to prevent a long delay between click and completion of the task. This is necessary, as it is harder to develop insight if one is constantly waiting

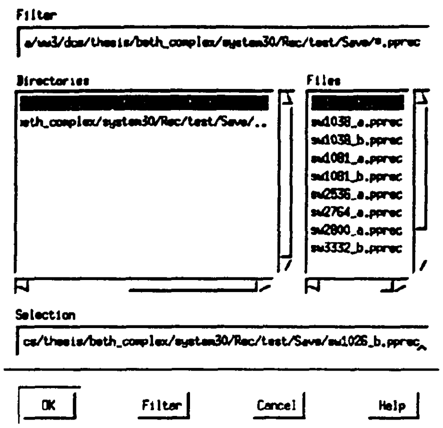

Figure 3-4: `vtrace` File Loading Dialog Box

on the machine to display data.

There are two methods for loading the files fast, one for `.pptrace` files and one for `.trace` files. Both are based on the same idea that the machine can scan the trace file for trace locations faster than it can parse the whole file at once. Once the locations have been calculated, loading the actual trace information causes minimal delay, so the system can do that when a specific drawing request is made. The file locations are read in using the C function `ftell()`, and are later used to move the file pointer to the proper location using the function `fseek()`.

When the `vtrace` viewer loads the `.trace` and `.rec` files, it stores all relevant information in an internal list of putative hits. Each putative hit element contains all of the information necessary to draw a trace – its type, its score, its start and end times, its conversation, its trace location in the trace file, and its list of truth labels. Most of this information is loaded from the other information files if they are available. If some of the optional information is not available, `vtrace` viewer simply does not include that information in the trace. The final step in loading the new conversation is to sort the internal list of putative hits. This is done using the `qsort()` function from the C library. `vtrace` viewer then gives this sorted list to the selection widget, from which the user can make a putative hit selection. This selection widget can be seen in Figure 3-5.

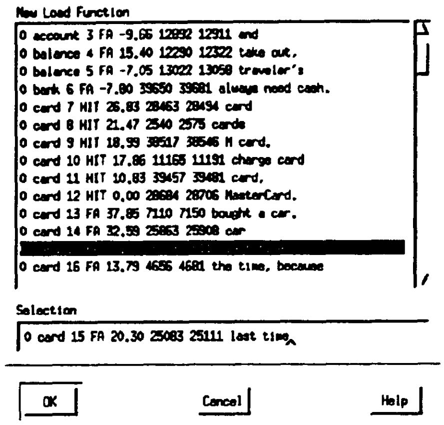

Figure 3-5: Putative Hit Selection Dialog

Drawing The Trace

When a putative hit is selected, `vtrace` will display the trace information on its drawing screen. Only the best path through the trellis is displayed. This is because for real models and data, there are too many model states and frames of data to show a complete trellis. For example, "credit card" is modeled with 10 triphones with three states each, and it often span 80 or more frames of data. Displaying the data as a full trellis of that size would make visualization difficult.

After a selection is made, the information from the selection widget is used to look up the specific putative hit in the putative hit list. Once this hit has been found, all the information needed to draw its trace is available. The file name and file position are used to open the trace file and `fseek()` places the file pointer at the proper location in the trace file. Model names, as well as the number of states, state occupancies, and per frame scores are all read in from the trace file at this time. From this information, most of the objects on the trace screen can be calculated. The model names, model topologies, state occupancies, normalized log likelihood score, and hypothesized labels (what the HMM system though had been said) are all included here. Information about truth has either already been loaded, or is assumed unavailable. A picture of this output can be seen in Figure 3-6.

Once all of this information has been acquired, the program is ready to begin calculating where the elements of the trace should be drawn on the screen. First, `vtrace`

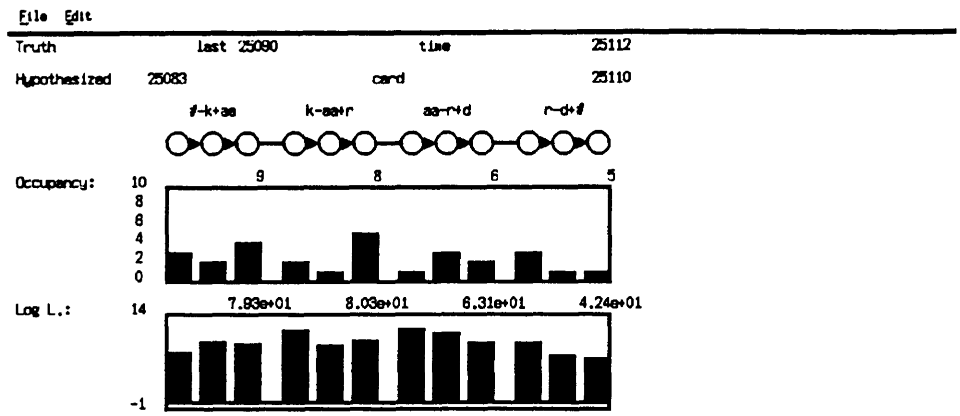

Figure 3-6: Output Window for vtrace with Example Putative Hit

calculates the vertical locations of all of the elements. Since there is no possibility of overlap here, the calculation is straight forward. Next, the horizontal locations of the models are calculated, one model at a time. Here, vtrace looks for places in the truth field and the hypothesized field where displaying labels based on their relative time positions would cause overlap. If such places are found, the models are spread out to prevent such overlap. After the model positions have been calculated, the information is used to calculate the locations of the bars in the occupancy and log likelihood bar graphs. The vertical sizes are calculated using the **xygraph** subroutine written by Richard Lippman. Two calls to **xygraph**, one for occupancy and one for log likelihood, finish the drawing.

3.3 Initial Insights

The goal of looking at the Viterbi backtrace for specific putative hits was to develop insight into why some hits scored lower than some false alarms. Several ideas were considered as a result of looking at the backtrace for the word "card" and its top few false alarms.

One reason for high scoring false alarms was that certain triphone models did not discriminate well between occurrences of the true triphone and the other triphones. One example involved the most costly false alarm for the "card" model: "car".

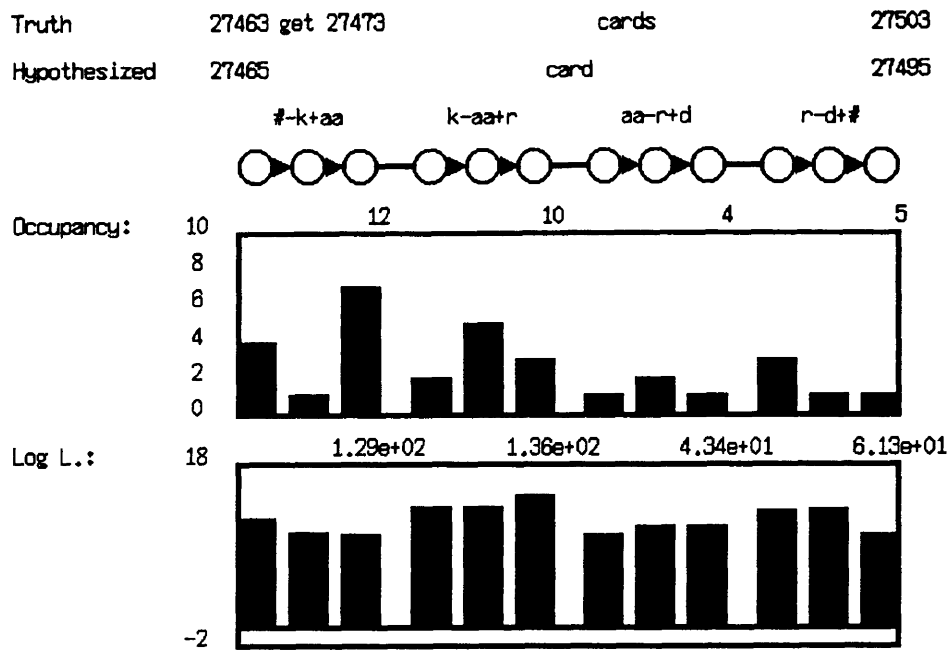

Figure 3-7: True Hit for "card", Score = 23.66

Compare the true hit in Figure 3-7 with the false alarm in Figure 3-8. For the false alarm, one would expect that the `r-d+#` model would not score the speech frames at the end of "car" as well as it did, because the /d/ sound did not occur. This would make the overall word score lower than the true hit scores. However, the `r-d+#` model did score the end of "car" reasonably well. This suggests that the `r-d+#` model does not discriminate as well as it should.

For other putative hits, `vtrace` would show strange patterns of state occupancies, as shown in Figure 3-9. In this example, the speech frames did not match up with the `#-k+aa` model very well. This did not affect the overall score because the corresponding duration for this model is much shorter than the normal duration for `#-k+aa`, as seen in Figure 3-7. The short duration prevented the expected accumulation of bad scores that should accompany segments of speech which do not match well against the model. Because the rest of the utterance matched with the rest of the model well, the utterance received a high scored. In this way, abnormal occupancy patterns can assist a false alarm to score well. These kinds of observations led to the idea that improving the modeling of durations in the hidden Markov models might help improve word spotting performance. However, developing such a test would be beyond

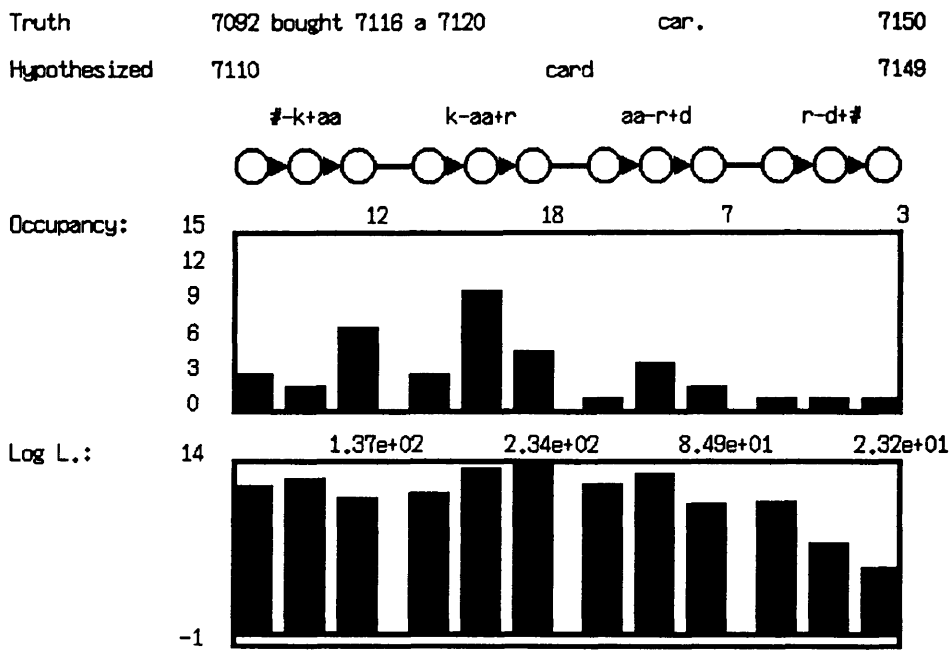

Figure 3-8: False Alarm "car" for "card" Model, Score = 37.85

the scope of this thesis, as it would have involved significant additions to both the training and testing software.

A second improvement idea for the word spotter was to limit the contribution any single model may have on recognition decisions. Analysis showed that certain phonetic models tended to dominate recognition, resulting in recognition decisions based on a subset of the phonemes rather than on the entire word. For example, word scores from the "card" model reflected the scores of the `k-aa+r` and `aa-r+d` models more than the scores of the `#-k+aa` and `r-d+#` models. This might be because the dominating models scored higher and matched more frames of data than the other models. One way to limit the contribution of these models might be to set an upper bound on the frame score of the model states. When the system calculates the state likelihood of a single frame of data, the system would limit the score if it is above the maximum threshold. A second way to limit model contributions might be to normalize the model contributions based on their occupancies. One scheme for occupancy normalization might be to recalculate the word scores based on an equal contribution from each state of each phone in the word model.

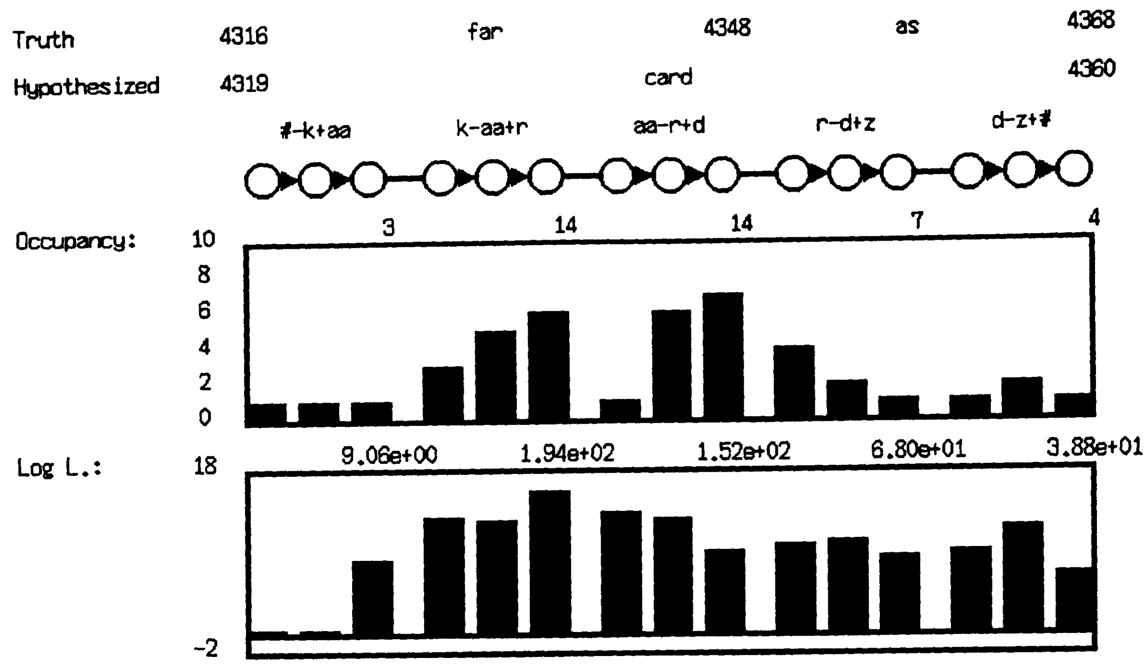

Figure 3-9: False Alarm "far as" for "card" Model, Score = 25.07

3.4 Secondary Test

After looking at several trace pictures, such as those shown in Figure 3-7 through Figure 3-9, it became apparent that the per state occupancies and per state normalized log likelihoods might be useful for discriminating false alarms from true hits. It seemed reasonable to run a secondary test, after the Viterbi decoder, to try to improve word spotting performance [12]. Because software existed to facilitate this type of test, such an experiment was run.

A multi-layer perceptron (MLP) classifier developed as part of the LNKnet software package was used as the secondary test [6]. The MLP input features were the per state occupancy and per state normalized log likelihood of 100 true hit examples of "card" and 181 false alarm examples of "card" from the training data, as shown in Figure 3-10. The training false alarms included all false alarms which scored higher than the lowest scoring true hit. The MLP then classified 43 true hits (as judged by the word spotter) and 97 false alarms (as judged by the word spotter) running on the testing data. The MLP took the per state occupancies and per state normalized log likelihood scores and removed from the putative hit list those hits judged to be false

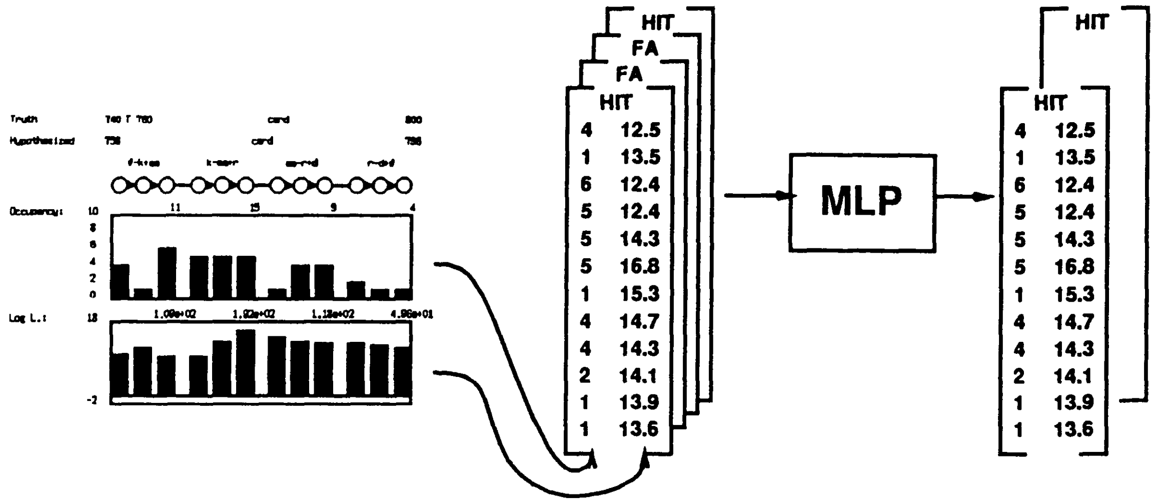

Figure 3-10: MLP Classification on Per State State Occupancies and Per State Normalized Log Likelihood

alarms. The secondary test did improve the performance of the system on the word "card", as the original FOM was 42.2 and the secondary test improved it to 48.5.

This test was run again for words other than "card". Unfortunately, to date no similar results have been achieved. This confirms previous experiences with secondary testing, which is that secondary testing can sometimes help and can sometimes have no effect [7]. Because of this information, and because this line of research was off the principle subject of the thesis, no major effort was put into developing this test.

3.5 Conclusion

The backtracing seems to be useful. It provides a way to make sure the system is decoding as expected. Also, several ideas about what can improve our word spotter have been formulated. Perhaps some of these ideas may lead to improvement in system performance. Additionally, as a result of work with the `vtrace` viewer program, a secondary test was inspired and yielded some promising results. Unfortunately, some of the ideas for improvement mentioned in Section 3.3 were deemed too time consuming to fit in to the scope of this thesis project.

3.6 Future Work

There are several ideas for future study related to the `vtrace` viewer. The ideas formulated from using the `vtrace` viewer could be implemented, and improvements to the actual `vtrace` viewer system can be made. The word spotter system improvements are listed in 3.3. The subject of `vtrace` viewer improvements is discussed below.

The `vtrace` viewer project has been taken over by Linda Sue Sohn, a full-time software engineer at Lincoln. Currently she is being guided by the author to implement improvement features for the `vtrace` viewer system. Some of the improvements already implemented in this effort include rewriting the GUI code to make it more efficient, as well as using the `qsort()` library function for loading and sorting putative hits quicker. Additional new features include an option to display several different traces at once, as well as a hook to an `xwaves` interface, which will allow a user to listen to a putative hit as well as to view the waveform and spectrogram of the putative hit with the word spotting transcription.

Chapter 4

Multidimensional Scaling

4.1 Introduction

A word spotting system compares speech segments to internal models for words and makes decisions based on how well those segments fit the internal models. In the process of improving such methods, it may be useful to provide pictures that show the positions of one model relative to another model or of a set of model centers relative to speech segments. For our case, this involves an interesting technical challenge. If speech were represented using two parameters, it would be possible to display speech segments as well as model centers on a two dimensional picture by taking one parameter as the X axis, and one parameter as the Y axis. However, in this system, speech is represented using two data streams, 12-dimensional cepstra and 13-dimensional delta cepstra. By using multidimensional scaling (MDS), the data's dimensionality can be compressed and meaningful two dimensional plots can be produced. Unlike `vtrace`, the MDS plots will not be exact representations of the data, but it is hoped that the meaning of the data will be retained in the scaling process.

Three experiments were run. The first experiment attempted to display the relationship between speech vectors from hits and false alarms to the Gaussian centers from the word models. The second experiment displayed the monophone set from the word spotter to try to determine those those sounds that may be difficult to distinguish given the data set. The final two experiments displayed the relationships between models which had been trained on different amounts of data in an attempt to determine the effects of limited training data on model location. Each of these experiments is discussed below.

4.2 Method

Multidimensional scaling provides a mechanism for projecting vectors in an N-dimensional space to a representation in lower dimension, while still maintaining relative distances between vectors [5]. As seen in Figure 4-1, there is a three step process for converting

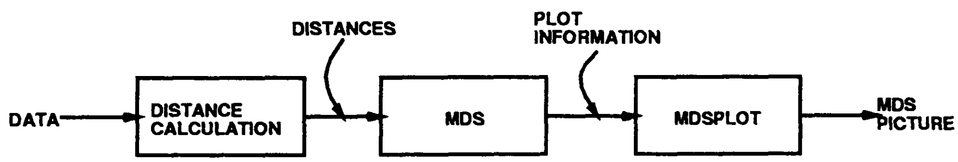

Figure 4-1: Three Step Process for MDS

raw data to an MDS plot. First, the distances between all of the data points in the N-dimensional space must be calculated and stored in a matrix. Second, the MDS program is called to produce a two dimensional representation of points related by the distance matrix. Finally, this picture is plotted using a suitable program. In this case it was a program written at Lincoln Laboratory called `mdsplot`.

4.2.1 Distance Calculation

The first part of the method shown in Figure 4-1 is the distance calculation. There are three important factors for choosing a distance metric for MDS. First, the distance must be inversely proportional to the nearness of the data points. It is also important that the same method can compute distances between all points in a single MDS picture. Finally, the chosen distance metric should relate in a meaningful way to the system being studied.

The two distance formulas detailed below were used to create the distance matrix described in Section 4.2. The Mahalanobis distance was chosen because of its similarity to the word spotter scoring method for the case of comparing model centers to speech vectors. Additionally, it provides a meaningful metric for distances between speech vectors as well as distances between model centers. The Bhattacharyya distance was chosen because of its usefulness in determining model confusion for other speech tasks [11]. Both of these metrics are described below.

Mahalanobis Distance for Gaussian Random Vectors

The first distance metric used for MDS was a weighted Euclidean distance, or Mahalanobis distance [3]. This is the Euclidean distance where each of the features are normalized to have variance of one. We can calculate the distance between two vectors, X and Y with common diagonal covariance Λ as follows:

$$X = \begin{bmatrix} x_1 \\ . \\ x_N \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ . \\ y_N \end{bmatrix}, \Lambda = \begin{bmatrix} \sigma_1 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & \ldots & 0 \\ 0 & 0 & \sigma_3 & \ldots & 0 \\ . & . & . & & . \\ 0 & 0 & 0 & \ldots & \sigma_N \end{bmatrix}.$$

$$d(X,Y) = (X-Y)^T \Lambda^{-1} (X-Y) \tag{4.1}$$

While the Mahalanobis distance is similar to the word spotter distance between a speech vector and a model center, there are some differences. First, the word spotting system implements a tied Gaussian mixture model (TGMM) probability density function with 128 model centers. Each center C_i has an associated covariance matrix Λ_{C_i}. So the distance between a speech vector X and a specific center C_i is as follows:

$$d(C_i, X) = (C_i - X)^T \Lambda_{C_i}^{-1} (C_i - X)[1]. \tag{4.2}$$

Because MDS requires a single metric for all distance calculations, the distance metric needs a single global covariance matrix Λ. So, the distance metric relating model centers to speech vectors for MDS is an approximation to the distance metric relating model centers to speech vectors used by the word spotter.

There is a second difference associated with relating the distance metric used by the word spotter to the Mahalanobis distance used here. For computational reasons, the word spotter does not use all 128 Gaussian mixture centers in its distance calculation. Rather, it chooses the five centers closest to the speech vector and averages their distances. The distance to the different state density functions is based on these five closest centers as shown below:

$$d(\lambda, X) = \log[\sum_{i=1}^{5} w_i \exp(d(C_i, X))], \tag{4.3}$$

where λ is the specific tied Gaussian mixture density function, X is the speech vector, C_1 through C_5 are the five closest centers to the speech vector, and w_1 through w_5 are the weights for these centers in this model. Because a consistent distance metric was desired for all states, and because useful pictures would include several different speech vectors, the Mahalanobis distance was used to relate the speech vectors to each of the top weighted centers for each state individually rather than jointly as in Equation 4.3.

A third difference is that whereas the word spotter actually uses the 12 cepstra and 13 delta cepstra values for each frame of speech, only the cepstra were used for MDS.

Bhattacharyya Distance for Gaussian Models

The second distance metric used for MDS was the Bhattacharyya distance. This distance is based on an estimation of the expected probability of error for two multimodal Gaussian probability density functions, and is not the exact distance between two Gaussian mixture models. However, it has proven useful for other speech processing problems, such as speaker identification using Gaussian mixture models [11].

For two probability density functions representing two classes, $p(x \mid \lambda_1)$ and $p(x \mid \lambda_2)$ with *a priori* class probabilities P_1 and P_2, the expected probability of misclassification is defined as

$$\epsilon = \int \min(P_1 p(X|\lambda_1), P_2 p(X|\lambda_2)) dX \tag{4.4}$$

and can be seen in Figure 4-2 [11].

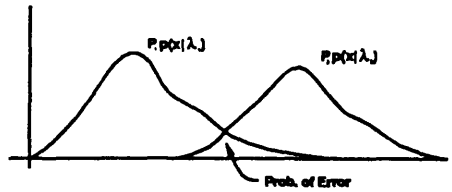

Figure 4-2: Depiction of the Probability of Error Between Two Class Distributions

Because the calculation in Equation 4.4 has no closed form for Gaussian mixture models, it is often approximated, such as with the Bhattacharyya bound [11]. The Bhattacharyya bound calculates an upper limit on ϵ [11]. For two Gaussian densities, $\lambda_1 = \mu_1, \Lambda_1$, and $\lambda_2 = \mu_2, \Lambda_2$, the Bhattacharyya bound is calculated as follows [11]:

$$\epsilon \leq \epsilon_b = \int \sqrt{\min(P_1 p(X|\lambda_1), P_2 p(X|\lambda_2))} dX = \sqrt{P_1 P_2} e^{-\rho} \tag{4.5}$$

where

$$\rho = \frac{1}{8}(\mu_2 - \mu_1)\Lambda^{-1}(\mu_2 - \mu_1)^T + \frac{1}{2}\log\frac{|\Lambda|}{|\Lambda_1||\Lambda_2|}. \tag{4.6}$$

The quantity ρ is known as the Bhattacharyya distance between densities [11]. It can be used to create a distance metric which relates two tied Gaussian mixture models, $p(x|\lambda_1)$ and $p(x|\lambda_2)$, where $p(x|\lambda_1)$ and $p(x|\lambda_2)$ are defined as

$$p(x|\lambda_1) = \sum_{i=1}^{M_1} p_i b_i(x), \tag{4.7}$$

$$p(x|\lambda_2) = \sum_{j=1}^{M_2} q_j a_j(x), \tag{4.8}$$

in which $b_i(x)$ and $a_j(x)$ are Gaussian densities. The distance used was

$$d(\lambda_1, \lambda_2) = -\log\left[\sum_{i=1}^{M_1} \max_{1 \leq j \leq M_2} \sqrt{p_i q_j} \exp -\rho(i,j)\right], \tag{4.9}$$

where $\rho(i,j)$ is the Bhattacharyya distance between b_i and a_j [11].

`HHDist`, a program to calculate this distance for HTK models, has been written by Beth Carlson, a researcher in the group. With a few minor changes, it was able to produce the distance output which MDS could convert into a two dimensional plot of the models.

The Bhattacharyya distance is not representative of any part of the word spotter training or testing. Both of these phases of word spotter operation maximize the likelihood of the data rather than maximizing the discrimination, therefore distances between models are never required. However, the distance between models certainly does effect performance. Therefore, the Bhattacharyya distance was employed to help gain intuition into the positioning of models.

4.2.2 Multidimensional Scaling Method

The pamphlet "Multidimensional Scaling" by Kruskal and Wish gives a basic overview of MDS [5]. The method transforms a high dimensional representation of data into a lower dimensional representation, while preserving the relative proximities of the points as dimensions are removed.

The first step in MDS is to compute the distances between all data points X, and store them in a matrix, Δ, shown in Equation 4.10.

$$\Delta = \begin{bmatrix} \delta_{1,1} & \delta_{1,1} & \dots & \delta_{1,N} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,N} \\ \cdot & \cdot & & \cdot \\ \delta_{N,1} & \delta_{N,2} & \dots & \delta_{N,N} \end{bmatrix} \tag{4.10}$$

X is the set of all data points x_i as shown in Equation 4.11.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \tag{4.11}$$

So, the distance between two points x_i and x_j would be calculated as in Equation 4.12, where δ is some meaningful distance function, such as those described in 4.2.1.

$$\delta_{i,j} = \delta(x_i, x_j) \tag{4.12}$$

Once Δ has been computed, all N of the data points are scaled using the following iterative process. First, they are placed in R dimensional space, where initially $R = N$, using a stress minimization process described below. Once the points have been placed, a dimension is removed, and the points are re-placed in this lower dimensional space. This process of removing one dimension and reconfiguring the points proceeds until the required dimension has been reached.

Points are placed in the scaled space by minimizing the stress between the Euclidean distance of the scaled space, d, and the proximity information in Δ. This is done in a four step process. First, $d_{i,j}$ is calculated for each i and j, based on their initial position, as in Equation 4.13, where $x_{i,r}$ is the rth dimension of x_i.

$$d_{i,j} = \sqrt{\sum_{r=1}^{R}(x_{i,r} - x_{j,r})^2} \tag{4.13}$$

Next, the "**f-stress**" of the current X placement is calculated, as seen in Equation 4.14, where $f(\delta_{i,j})$ is the objective function which relates proximity, δ, to distance in the space, d.

$$\textbf{f-stress}(\Delta, X, f) = \sqrt{\frac{\sum_i \sum_j [f(\delta_{i,j}) - d_{i,j}]^2}{\sum_i \sum_j d_{i,j}^2}} \tag{4.14}$$

The **stress** of the current Δ and X configuration is calculated by finding the best objective function:

$$\textbf{stress}(\Delta, X) = \min_{argf}(\textbf{f-stress}(\Delta, X, f)) \tag{4.15}$$

Finally, the best fit $\hat{X}$ for the proximity matrix Δ involves choosing the X which minimizes the stress of the system, as seen in Equation 4.16.

$$\mathbf{stress}(\Delta, \hat{X}) = \min_{argX}(\mathbf{stress}(\Delta, X, f)) \tag{4.16}$$

Actually, the above process, while useful for defining the best fit, does not provide a good way to calculate $\hat{X}$ given Δ. For this, Kruskal and Wish used the method of steepest descent to find $\hat{X}$. An analogy of this method is reproduced here. Imagine a three dimensional graph, where x and y are dimensions of the configuration, and z plots the stress. Essentially, the method involves dropping a ball on this terrain at any point, and letting it find the lowest point it can by following the line of steepest descent. This method has one main drawback. It is possible that the calculation of $\hat{X}$ will produce a local minimum, although the above method of starting with $R = N$ makes it unlikely. Producing a local minimum is undesirable, but there is no way to be absolutely sure it does not happen, short of calculating all possible stresses for all possible positions. According to the pamphlet, however, becoming trapped in a local minimum is rare.

For this thesis, a 1973 version of the MDS program was used, because it was the only MDS program the group had. Because the program used was an older version, it could only scale up to 62 points at a time. It is realized that newer programs are available which do not have such a limit, but they were not available for this thesis.

4.2.3 Output: mdsplot

The output of the multidimensional scaling program can be converted into standard X and Y coordinates, and simple plotting packages exist already to display the picture. However, to facilitate the display of speech models and speech segments, a special plotting program was created to display MDS type plots. `mdsplot`, developed by Linda Sue Sohn, takes as input simple plotting commands (such as `Circle (x, y, r, color)`) and draws the MDS picture.

The final pictures are graphs of the data elements in two dimensions. Because the points are placed by using an iterative process which minimizes the difference between actual distances and the displayed distances, the meaning of the two dimensions in the final picture is unclear and changes from experiment to experiment.

4.3 Experiment 1: Relating A True Hit and a False Alarm to a Model

The first MDS experiment compared a high scoring false alarm to an example of a true hit to see if significant differences could be seen. As mentioned in Section 2.2.3, "card" was the word on which most of this analysis took place. The highest scoring false alarm (i.e. the word which scored best in the card model) was an instance of the word "car". In fact, for figure of merit calculation, "car" factors into the poor performance of "card" much more than the rest of the false alarm words combined. One would expect little difference in the beginning parts of these words, as "car" and "card" are initially the same. In contrast, it is expected that the lack of a /d/ sound in for "car" would result in divergent tracks at the end.

The six top examples of "car", as defined by their score from the "card" model, were paired up with the six top examples of "card". These pairs, in addition to the Gaussian centers from the triphones for the word "card" were run through the MDS system. The vectors from the words were the 12 cepstral coefficients of each frame of data. For the models, which were tied Gaussian mixture models with 128 model centers per model state, only the two or three centers with the highest weight for each of the three model states were chosen. The Mahalanobis distance metric was used. A typical result can be seen in Figure 4-3. The circles represent the different model centers and are included only for display purposes. The center labels describe which specific center each was, as well as which models the specific center represents. For example, the model label "d.35" refers to the 35th Gaussian center which represents one of the centers with highest weight among the three states in the `r-d+#` triphone. The two lines show the progression of cepstral observation vectors for the two words scaled into two dimensions. The solid line represents a true hit, while the dashed line represents a false alarm from a different speaker. The labels describe which word the

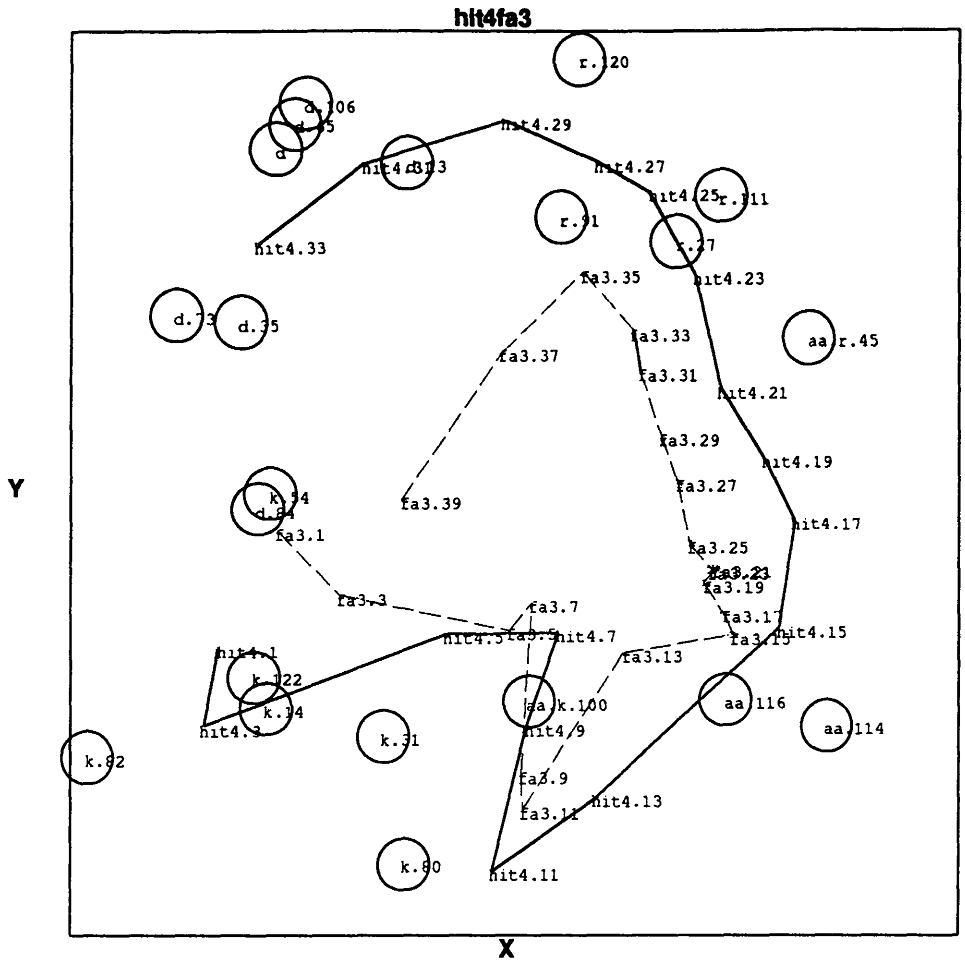

Figure 4-3: MDS Output for Experiment One: Hit vs. False Alarm

data came from as well as which specific frames are being shown. For example, the data frame labeled "hit4.23" means that the data point came from the 4th highest scoring hit from the word spotter's "card" model, and this was data frame 23 for that word. In comparison, "fa3.35" means that this data point came from the 3rd highest scoring false alarm for the "card" model, and this was data point number 35.

The words tracked along the model centers for the /a/ and /r/ sounds and diverged at the /d/ sound as expected, but the tracks also diverged at the /k/ sound where it was expected that they would be more similar. Although the /k/ sounds for "car" and "card" were different, they were both within the spread of the **#-k+aa** model's centers. The last few frames of the false alarm are different, however, and the data points "fa3.37" and "fa3.39" do not appear within the spread of the **r-d+#** model's centers. This suggests that while both tokens of the /k/ sounds appear to be close to their model centers, the false alarm token of the /d/ sound appears to be far from its model centers. This result was interesting because this token of "car" scored well in the "card" model, even though its last few frames were far away from the **r-d+#** model. This suggests that the **r-d+#** model did not affect scoring very much. It also suggests that there are differences between the ends of "card" and "car" that can potentially eliminate "car" as a false alarm.

A second example from this experiment is the case of scaling two true hits, shown in Figure 4-4, which can be compared with the hit vs. false alarm picture of Figure 4-3. This figure differs from the hit vs. false alarm case as expected because the

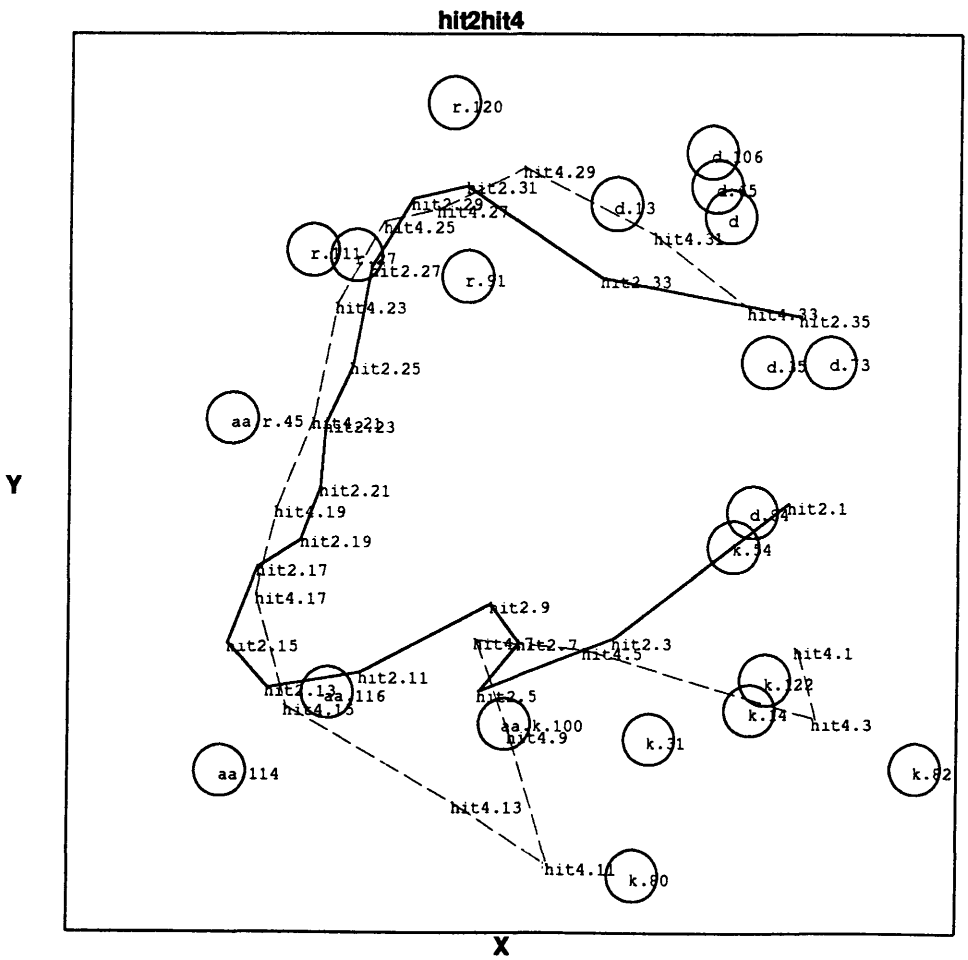

Figure 4-4: MDS Output for Experiment One: Hit vs. Hit

end sounds, where there really is a /d/ for both examples, do not diverge. Again, however, one can see that some of the frames from the /k/ sound and the /aa/ sound were not as close to each other as one might expect. For example, "hit4.11" was not close to the track of the other true hit as expected. However, it was close to the "k.80" model center suggesting again that this wide variation may be normal for the /k/ sound. Another, difference between Figure 4-3 and Figure 4-4 is that the similar data points are not in similar locations. The word tracks in Figure 4-3 proceeded counterclockwise while the tracks in Figure 4-4 proceeded clockwise. This is because the MDS procedure not guaranteed to produce similar pictures for similar data, and it emphasizes that caution must be used when interpreting MDS pictures.

Figure 4-5 is an alternative hit vs. false alarm picture. This shows one example

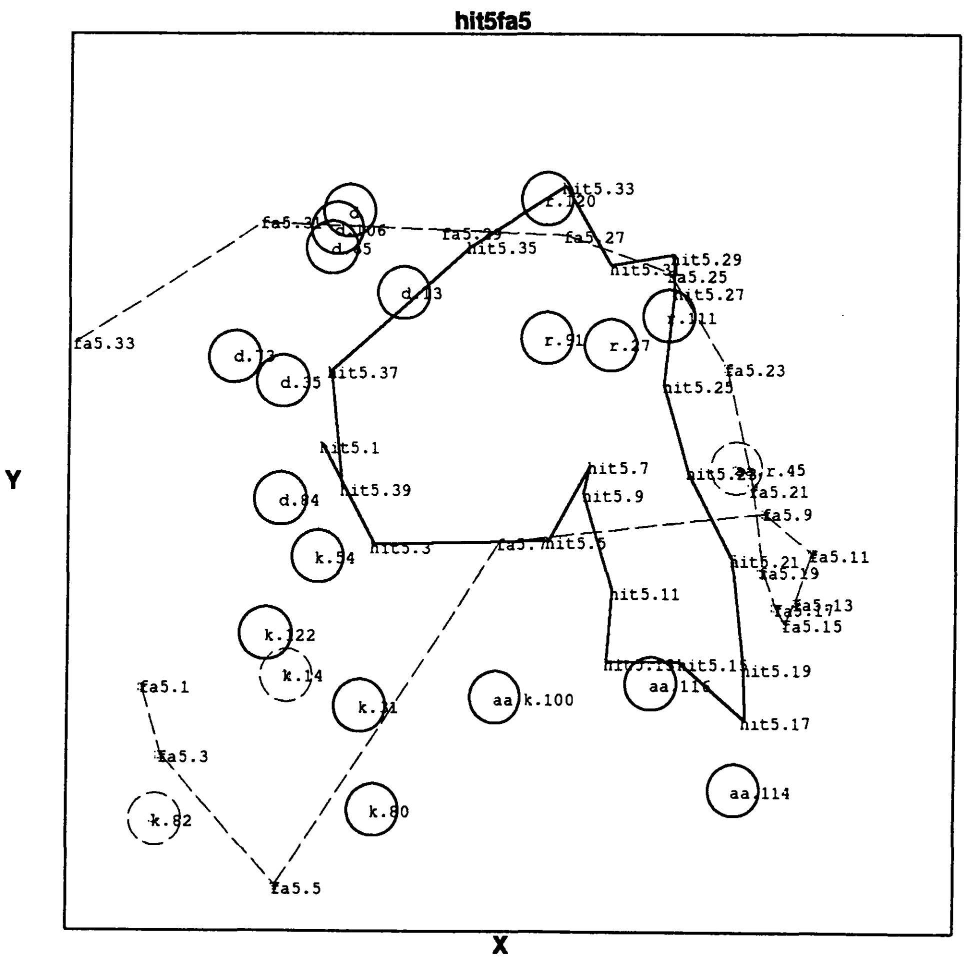

Figure 4-5: MDS Output for Experiment One: Alternative Hit vs. False Alarm

where the tracks for "car" and "card" were very dissimilar, and the plot is a bit more difficult to interpret. However, if we look at the actual vector distances in Table 4.1, we see that the MDS plot is a relatively accurate representation of the data. For example the vectors "fa5.1" and "fa5.3" had a Mahalanobis distance of 3.3

Table 4.1: Mahalanobis Distances for Speech Frames

	fa5.1	fa5.3	fa5.5
hit5.1	4.9	5.6	5.8
fa5.1	0.0	3.3	7.1
fa5.3	3.3	0.0	5.3

whereas "hit5.1" and "fa5.1" had a Mahalanobis distance of 5.6. These distances are reasonably reflected in the MDS picture.

4.4 Experiment 2: Clustering Models

Although the distance between the models is never used explicitly in the word spotter training or testing, it does relate to performance. In this experiment, the Bhattacharyya distance described in Section 4.2.1 was used to create an MDS plot of the monophone set from the word spotter. It was hoped that such a plot would provide insight as to which models might cause confusion for the system.

As can be seen in Figure 4-6, most of the models were clustered as one would expect by phonetic classification, while a few models seemed to be misplaced [14]. Some of the phones in unexpected locations include the **dx** model and the **k** model

Table 4.2: Bhattacharyya Distances for Monophones

	t	p	b	s	th	f
dx	14.7	14.4	14.0	14.0	13.7	13.5
k	6.7	4.9	8.6	9.2	7.5	7.6
b	7.1	5.3	–	10.4	9.0	9.7

(both expected near the other stops). The **dx** model is not placed near the other stops as expected. However, the Bhattacharyya distances to these models shown in Table

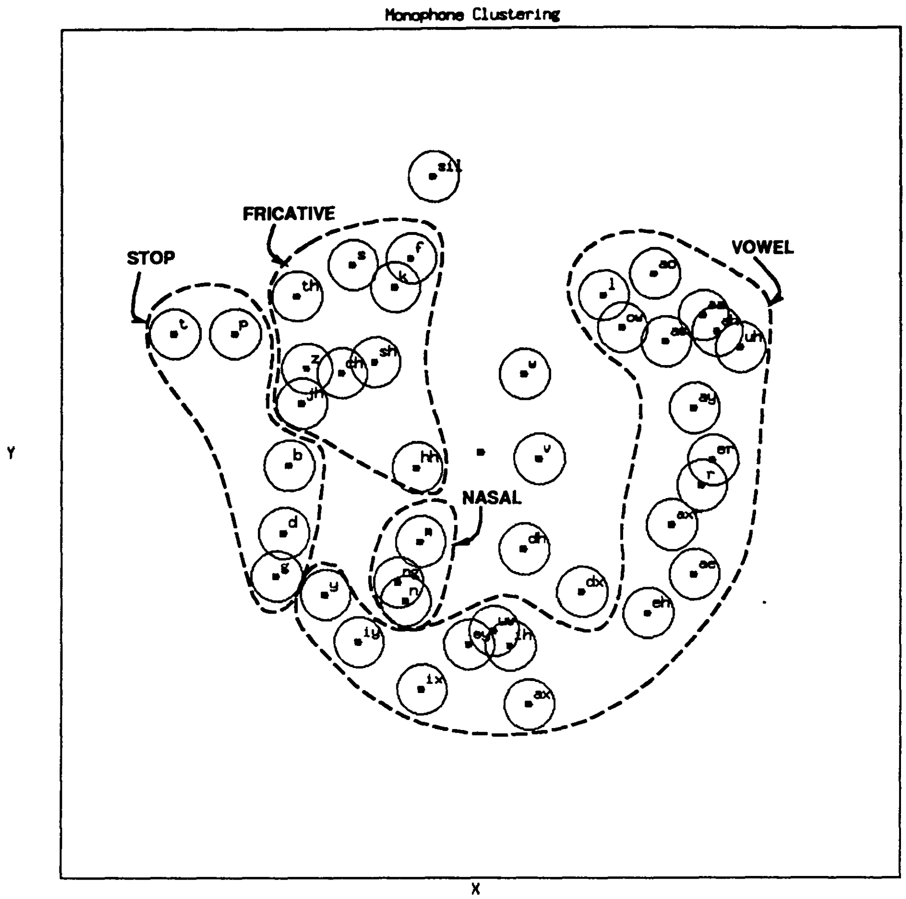

Figure 4-6: MDS Output for Experiment Two

4.2, support such a placement, as they show the **dx** model to be different from the other stop phones. In contrast, the **k** model is surrounded by the fricative models **th**, **s**, and **f**, and the Bhattacharyya distances, again in Table 4.2 do not support this, as **k** is closer to **t** and **p** than to **s**, **th**, and **f**. These two examples can be compared to the **b** example where MDS was successful. The **b** model is closer to the **t** and **p** models than to the fricatives, and the MDS picture reflects that. This shows that while MDS can assist in viewing data which are otherwise difficult to display, results must be checked against the original data.

4.5 Experiment 3: Amounts of Training Data

A final experiment involved exploring the differences between models which had been trained on different amounts of the training data. Because the Switchboard runs had been made using models trained on various amounts of data, it was desirable to see what affect varying the amount of training data has on model positions. Four model sets had been trained: set 1 had been trained with 100% of the data; set 2 had been trained with 75% of the data, set 3 had been trained with 50% of the data, and set 4 had been trained with 25% of the data.

Because of the 62 model limit imposed by the MDS system described in Section 4.2.2, two different experiments were run. In the first one, all of the monophones from the fully trained set were multidimensionally scaled. For some of the monophones (`uw`, `ng`, `er`, `dx`, `aa` and `sh`) the models from each training set were included. The results can be see in Figure 4-7. The circles are used to point out where the model locations actually are. The labels with no numeric extension are from training set number 1. The models with a numeric extension are from the partially trained model lists. A small segment of Figure 4-7 is shown in Figure 4-8. Three particular examples are traced, `uw`, `ng`, and `er`. Here, it seems as though the models are exchanging positions relative to each other with for each new data set. This may be explained by looking at the distances in Table 4.3. The distance between `ng.2` and `uw.4` is larger than

Table 4.3: Bhattacharyya Distances for Partially Trained Monophones

	ng	ng.2	ng.3	uw.4	m
ng	–	8.6	9.5	9.2	6.2
ng.2	8.5	–	7.6	9.0	9.5
ng.3	9.5	7.6	–	7.5	8.8

the distance between `ng.2` and `ng`, but the MDS plot in Figure 4-7 places `ng.2` and `uw.4` very close while it places `ng` much farther away. This shows that the scaling procedure can produce misleading pictures.

The second experiment involved displaying all training sets for all models. Because

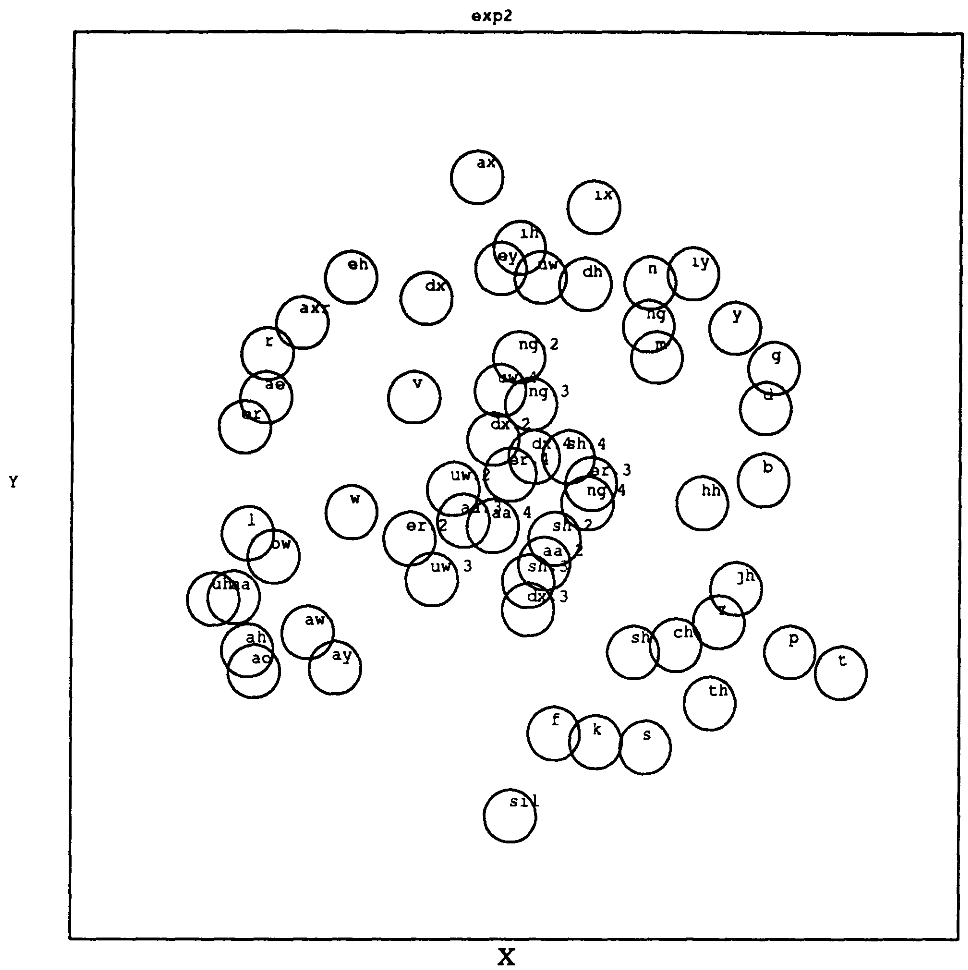

Figure 4-7: Output for MDS Experiment 3: All Monophones

this experiment involved many data points, the models were divided into three groups, based on the experiment in Section 4.4. Figure 4-9 displays one of the three MDS plots, although similar results were seen in the other two pictures as well. Figure 4-10 shows a subset of the information of Figure 4-9 which is easier to analyze. It shows that while some models seem to cluster apon a particular location as the training increases, others seem to reverse locations in this two dimensional picture. For example, it appears as though the `k` model is converging on a specific point, as the distances between model positions are decreasing as the amount of training data increases. It also appears as though the `t` model is not converging on any point relative to the other models. However, the picture is not representative of the data, shown in Table 4.4. The closest model to `k` is `t` according to the data, whereas the

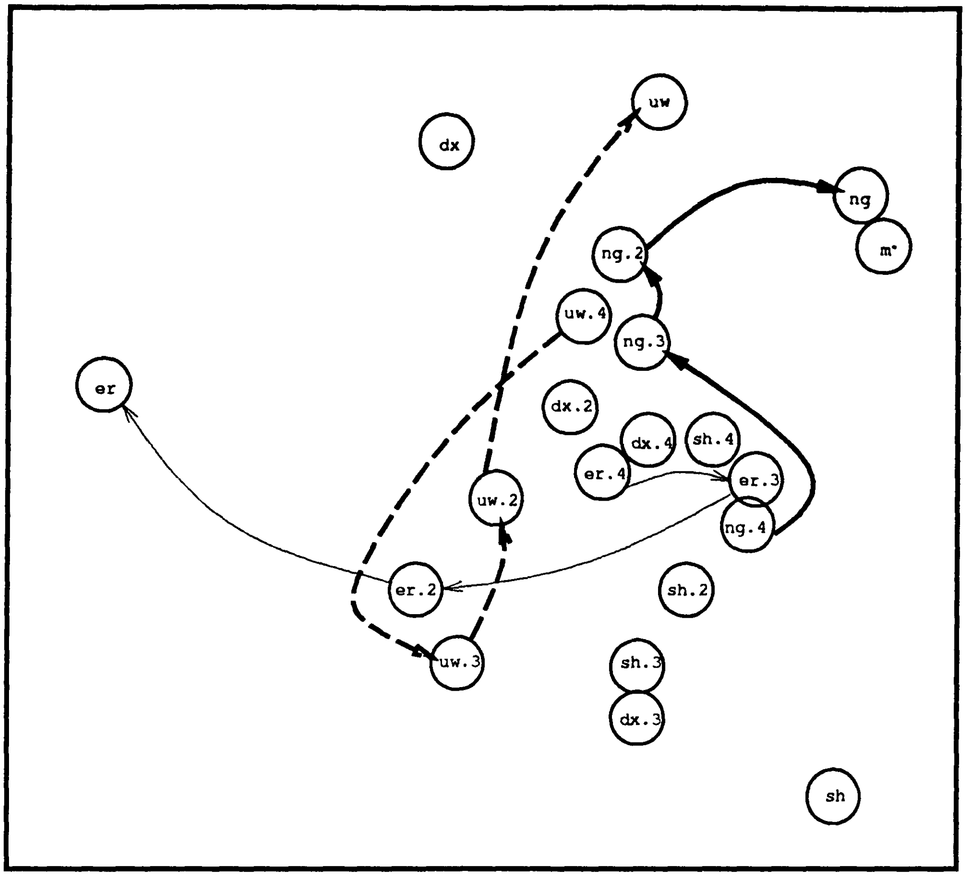

Figure 4-8: Partial Output of Experiment 3: All Monophones

closest model to k is k.2, according to the picture. The rest of the models follow the same trend, suggesting that in this case the picture is not an accurate representation of the data.

4.6 Conclusion

The MDS method seems to provide some useful insights for speech data. The first experiment attempted to demonstrate the difference between true hits and high scoring false alarms resulting from "car". For most of the example false alarms, this could be seen using the MDS method. However, MDS plots which were ambiguous show that the trends must confirmed up with further analysis of the original data.

The second experiment demonstrated that this method allows visualization of groups of models which might be confused for each other. The MDS picture placed

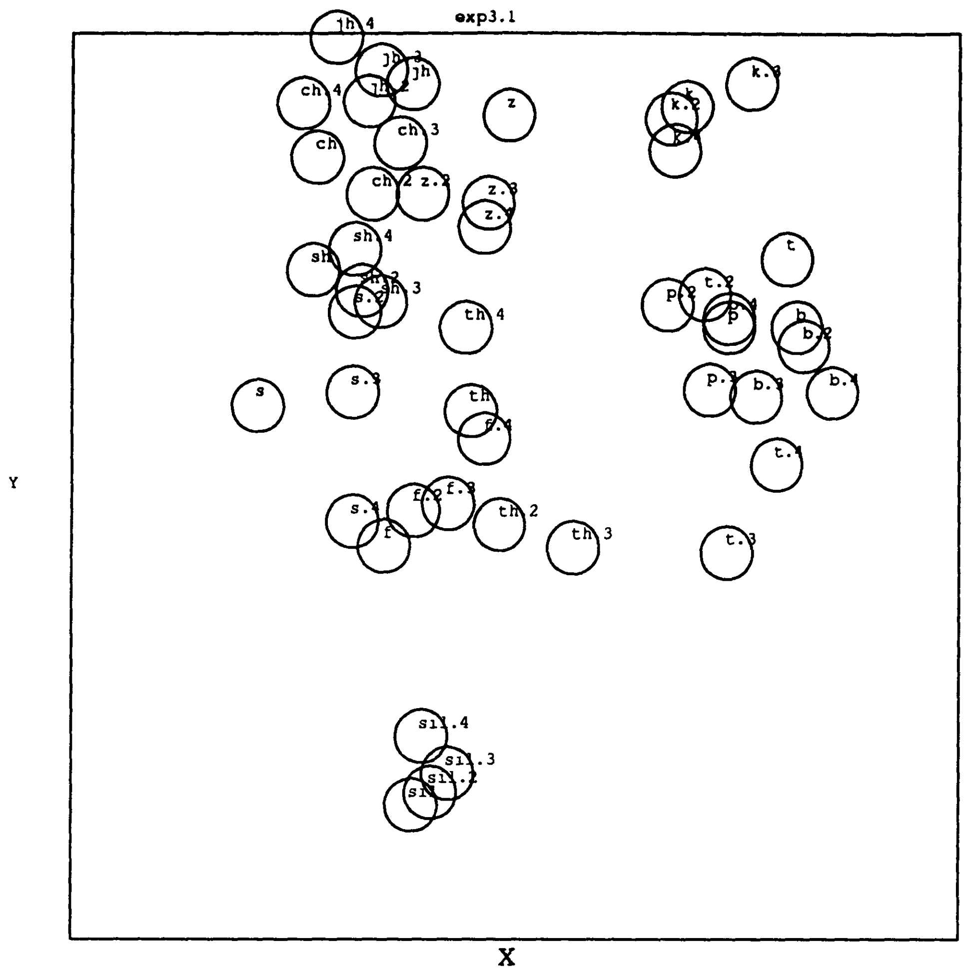

Figure 4-9: MDS Experiment 3: Partial Monophone List

most monophones in a reasonable way relative to each other. However, a few monophones seemed to be misplaced.

The third experiment demonstrated the effects of training on the confusedness between certain models. As more training data were used, certain models appeared to converge on a point, suggesting that these models may not be affected by the limited training data. However, on closer examination, the pictures seemed inconsistent with the true distances, leaving doubt regarding the ability of MDS to accurately represent the type of data.

In general, this method is only as good as the metrics behind it. There is no new information about point by point comparisons in data displayed using MDS. Additionally, the use of MDS adds an element of error due to the nature of its scaling

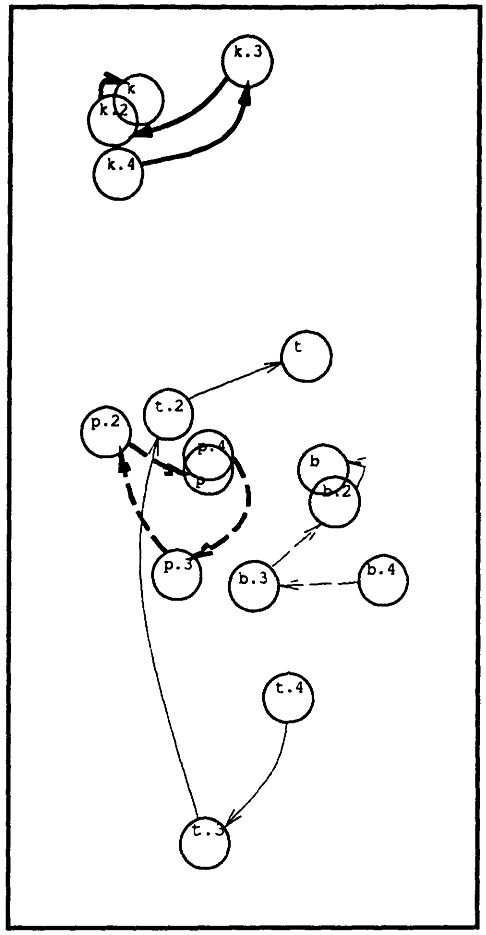

Figure 4-10: Partial Output of Experiment 3: Partial Monophone List

process. The overall conclusion is that the benefit of MDS is its ability to show high dimensional data on two dimensional plots, but it cannot be used blindly.

4.7 Future Work

The MDS software was written in 1973. As a result, the algorithm to perform MDS scaling can process a maximum of 62 points. It was most useful to look at points which had been scaled in a single invocation of MDS, and we would have liked to have added more than 62 points for some comparisons. New software that could handle more than 62 points should be installed at Lincoln Laboratory.

Another limitation of this method was the lack of a way to relate the radii of the circles to the actual distances between points. The circles ended up being nothing

Table 4.4: Bhattacharyya Distances for Monophones

	k	k.2	k.3	k.4	t	t.2	t.3	t.4
k	–	7.9	9.5	10.5	6.7	9.3	8.9	10.6
k.2	7.8	–	9.3	9.2	8.7	6.8	9.8	8.9
k.3	9.5	9.3	–	8.2	10.3	10.0	8.2	9.0
k.4	10.5	9.2	8.2	–	11.6	9.7	10.2	8.1
t	6.7	8.7	10.3	11.6	–	9.9	9.6	10.7
t.2	9.3	6.8	10.0	9.7	9.9	–	10.4	9.9
t.3	8.9	9.8	8.2	10.2	9.6	10.4	–	10.0
t.4	10.6	8.9	9.0	8.1	10.7	9.9	10.0	–

more than meaningless artifacts for display, rather than useful instruments for analysis. It was hoped that the radii of the circles for model centers would possibly define a region where speech segments would be strongly classified by the models. Definition of strong classification would be subject to the needs of the experiment. The inability to translate radii from the pre-scaled domain into the post-scaled domain prevented such analysis.

Additionally, while the experiments in Section 4.4 and Section 4.5 explored monophones, the word spotter uses triphones. Therefore, it would be a good idea to perform the same experiments using triphones.

Chapter 5

Conclusion and Future Work

The exploratory data analysis provided some insights into an HMM word spotting application. The backtrace viewer shed insight into state occupancy. This led to a secondary test which could improve the figure of merit of the word spotting system. The MDS technique showed how a word progressed through the different models. It was able to show the clustering of different model centers and models. It also could show how models move about relative to each other as they are trained. Some ideas for future work are discussed below.

5.1 Improvements on the HTK Word Spotter

First, adding a better duration model to HTK might improve word spotter performance. Data analysis suggested that odd paths through the models were in some way responsible for poor word spotter choices, either by scoring true hits too poorly or by scoring false alarms too well.

Second, it seemed clear that some models did not capture the full complexity of the sounds they were trying to model. From the MDS analysis, it seemed that the models were not converging on a specific point in the space. This suggests that the architecture of the models was insufficient. Perhaps different model architecture would be more trainable for the data.

Third, the MDS analysis also suggested that certain sounds were difficult to distinguish. Adding a system to the HTK word spotter which could notice when a sound was possibly within an ambiguous group and then performing some additional analysis might assist in distinguishing between these easily mistaken sounds.

5.2 Development of New Techniques

The development of two data analysis techniques and some of the insights gained by using them suggests that this general field of study should be explored further. In the proposal for this thesis, additional data analysis techniques had been suggested. Perhaps implementing these or other techniques and using them may yield additional insight to the problem of word spotting.

5.2.1 Testing State PDFs with Observations

One way to compare the training data to the testing data would be to plot only the $b_j(o_t)$ versus t for all j (see Figure 5-1 for an example). Recall that $b_j(o_t)$ is the probability that observation o_t was produced by state j. This could provide insight as to how the specific model states compared to the series of sounds making up the word. This graph could be time-indexed with a spectrogram of the actual waveform

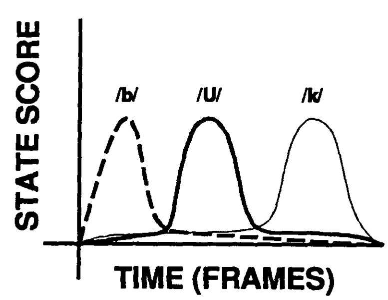

Figure 5-1: State Score vs. Frame Index

to compare $b_j(o_t)$, the state probability scores, with o_t, the observations.

Referring to the book example, one would expect that the probability the speech was produced by the b state would be highest during the time the /b/ phoneme was produced, and similarly for the U and k states for the /U/ and /k/ phonemes respectively. In this way, it would be possible to evaluate the state models for the phonemes of the word represented by the HMM in question.

5.2.2 Gaussian Mixture Analysis

As mentioned in the introduction to HMMs, the output of a hidden Markov model is determined by state-dependent probability density functions. To make useful probability density functions describing the speech observations, assumptions have to be made about the distribution of the speech vectors (e.g. are they modeled accurately by Gaussian mixture densities). As is demonstrated in Figure 5-2, these assumptions

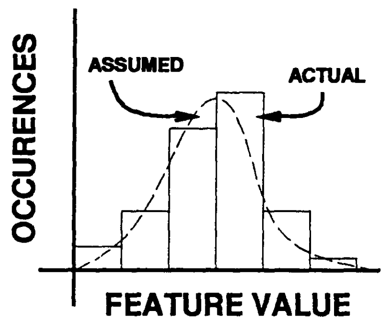

Figure 5-2: Real vs. Modeled Feature Distributions

can be checked by plotting the assumed distribution of feature vectors against the actual distributions for both the training data as well as the testing data. This could ensure that the data for a specific state's model fits the overall probability density function implementation. If this were not the case, the model's probability density function could be altered to reflect that. Additionally, states could be split if they are found to model in one node what could be better modeled in two. In this way, it may be possible to determine when an improper number of states has been used in specifying the HMM architecture.

Bibliography

[1] Beth A. Carlson. *A Projection-Based Measure for Automatic Speech Recognition in Noise.* PhD thesis, Georgia Institute of Technology, November 1991.

[2] Alvin W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill Book Company, 1967.

[3] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[4] J. J. Godfrey, Edward C. Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. *ICASSP '92 Proceedings*, 2:517–520, 1992.

[5] Joseph B. Kruskal and Myron Wish. *Multidimensional Scaling*, volume 07-011 of *Quantitative Applications in the Social Sciences.* Sage Publications, 1978.

[6] Richard P. Lippmann, Linda Kukolich, and Elliot Singer. LNKnet: neural network, machine-learning, and statistical software for pattern classification. *The Lincoln Laboratory Journal*, 6(2):249–268, 1993.

[7] Richard P. Lippmann and Elliot Singer. Hybrid neural-network/HMM approaches to wordspotting. *ICASSP '93 Proceedings*, 1:565–568, 1993.

[8] Jeffrey Neil Marcus. *Word and Subword Modeling in a Segment-Based HMM Word Spotter Using a Data Analytic Approach.* PhD thesis, Massachusetts Institute of Technology, September 1992.

[9] Douglas B. Paul. Speech recognition using hidden Markov models. *The Lincoln Laboratory Journal*, 3(1):41-62, 1990.

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

[11] Douglas Reynolds. A Gaussian mixture density distance measure. Internal Memo to Group 24, MIT Lincoln Laboratory, April 1993.

[12] J. R. Rohlick, P. Jeanreaud, K. Kg, H. Gish, B. Musicus, and M. Siu. Phonetic training and language modeling for word spotting. *ICASSP '89 Proceedings*, 2:459–462, 1989.

[13] J. Robin Rohlick, William Russell, Salim Roukos, and Herbert Gish. Continuous hidden Markov modeling for speaker-independent word spotting. *ICASSP '89 Proceedings*, 2:627–630, 1989.

[14] Douglas O' Shaughnessy. *Speech Communication, Human and Machine.* Addison-Wesley Publishing Company, 1987.

[15] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley Publishing Company, 1977.

[16] Steve J. Young and Phil C. Woodland. *HTK Version 1.4.* Lynxvale Limited, 20 Trumpington Street, Cambridge, England.