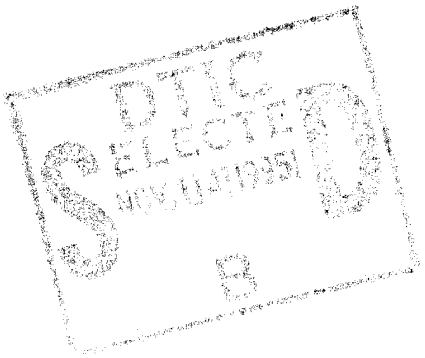
ESC-TR-94-110

Technical Report
1009

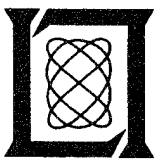
Graphical Analysis of Hidden Markov Model Speech Recognition Experiments



D.C. Seward IV
M.A. Zissman

25 October 1995

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



Prepared for the Department of Defense under Air Force Contract F19628-95-C-0002.

Approved for public release; distribution is unlimited.

19951113 067

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of Defense under Air Force Contract F19628-95-C-0002.

This report may be reproduced to satisfy needs of U.S. Government agencies.

The ESC Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tutungian
Administrative Contracting Officer

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission is given to destroy this document when it is no longer needed.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

GRAPHICAL ANALYSIS OF HIDDEN MARKOV MODEL SPEECH RECOGNITION EXPERIMENTS

D.C. SEWARD IV
M.A. ZISSMAN
Group 24

TECHNICAL REPORT 1009

25 OCTOBER 1995

Approved for public release; distribution is unlimited.

LEXINGTON MASSACHUSETTS

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 multidimensional scaling display. These tools were built using the HMM Toolkit. Use of the Viterbi backtrace tool provided insight that eventually led to improved recognition performance.

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

ACKNOWLEDGMENTS

We would like to thank Dr. Victor W. Zue for his guidance. His experience and knowledge were invaluable assets in this endeavor.

Many thanks also go to Group 24 at MIT Lincoln Laboratory, especially to those with whom many ideas were discussed at length and many problems associated with this work were solved: Jerry O'Leary, Beth Carlson, Richard Lippmann, Doug Reynolds, Terry Gleason, Charles Jankowski, Eric Chang, and Linda Sue Sohn.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

LIST OF TABLES

1. INTRODUCTION

Hidden Markov models (HMMs) are powerful tools for performing pattern recognition [1]. They are especially useful for speech recognition and word spotting, but that power comes with 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 to gain insight into successes and failures are not as well developed. This report describes data analysis tools that are integrated into an existing HMM software package and assist in the analysis of HMM experiments.

1.1 The Problem of Examining Experiments

The Hidden Markov Model Toolkit (HTK) [2] used at Lincoln Laboratory is a set of libraries and programs designed to facilitate creating, running, and evaluating HMM experiments. These experiments are evaluated mainly on the basis of recognition performance, and intermediate data analysis is typically not performed. Although HTK can output vast amounts of diagnostic information, there is no method for easily interpreting this information. So, while HTK simplifies creating and running HMM experiments, there exists no set of tools that eases data analysis.

1.2 Exploratory Data Analysis

One of the goals of data analysis is to open up a set of data so that it can tell its own story [3]. After creating unbiased pictures from diagnostic data, a researcher can gain a better understanding of what the system is actually doing. As this understanding improves, one should be more easily able to formulate ideas about current system limitations and areas for possible improvement. Therefore, one important goal when creating data analysis tools is to find techniques that are general and unbiased enough to permit widely different interpretations of the data.

The goal of any analysis is to improve its subject. A prototypical data analysis process [4] follows: First, run the system. Next, study intermediate data using data analysis techniques; suggest and implement improvements based on a better understanding of the problem and the system's actual implementation. Finally, evaluate any new design decisions to determine if they produce the desired results.

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 [5]. Rather, the system must strive for unbiased analysis. Second, the techniques should display useful information. If the data displayed are too general for meaningful analysis, the technique fails. Third, any implemented systems should be easy to use. Nothing is gained if a data analysis technique has been developed that no one can use [3]. 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

Two data analysis techniques are described. One produces the Viterbi backtrace for the putative hits of a recognition experiment (see Section 3) and includes information such as word model design, truth, and state occupancy, as well as the per state average of the normalized log likelihood scores. The other technique uses existing (and creates new) software to produce two-dimensional pictures of multidimensional data (see Section 4). Additionally, a few examples of data analysis experiments were run and evaluated.

2. BACKGROUND

2.1 Introduction to Hidden Markov Models

HMMs are useful for modeling speech production and for implementing speech recognition systems. However, the complexity of the models is as much a burden as an asset when it comes to implementing the training and testing phases of a speech recognition system. To combat this problem, many researchers at Lincoln Laboratory use HTK to run 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 speech processing system, 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 [5]. 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 1, the states of this model are `B`, or Burger King state, and `M`, or McDonald's state. Because this family has never eaten at either McDonald's or Burger King, they have only advertising and word of mouth to determine their initial choice. This situation 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 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 its output.

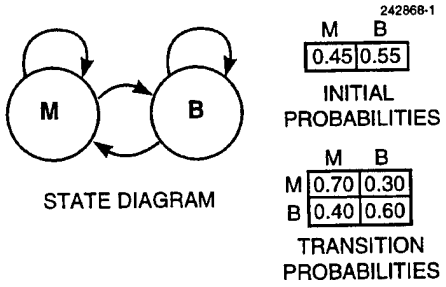


Figure 1. Example of a simple Markov model.

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

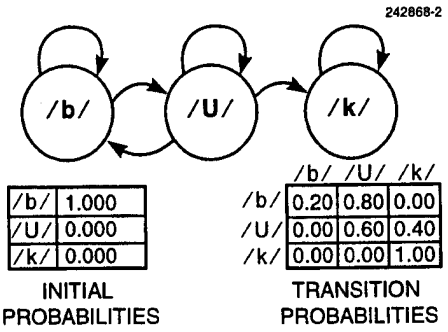


Figure 2. Example of a simple word Markov model.

Both these Markov models have produced output from which the state can be uniquely determined. For example, when eating at McDonald's, one is in state M. A *hidden* Markov model adds the ability of a state to produce output probabilistically. Perhaps being in state M means an 85% chance of eating in McDonalds and a 15% chance of eating at Burger King. For this case, the state sequence cannot be uniquely determined from the output in this case, thus it is referred to as "hidden." An HMM produces observations according to a state-dependent probability density function [6]. Given this form, HMM users must address three important problems [1]:

1. Given the observations $O = O_1, O_2, ..., O_T$, and the model M, how is $Pr(O|M)$ computed?
2. Given the observations $O = O_1, O_2, ..., O_T$, how is a state sequence $I = i_1, i_2, ..., i_T$ chosen that is optimal in some meaningful sense?
3. Given training observations $O = O_1, O_2, ..., O_T$ that are assumed to have come from a Markov process, how are the optimal model parameters calculated?

The solution to these three problems is described next.

Consider modeling an isolated word using an HMM, as illustrated in Figure 3. As before, there is a vector of initial state probabilities and a matrix of state transition probabilities. However, now the output is a probabilistically determined observation vector. The observation vectors might

be spectral envelope or low order cepstral generated by the probability density function of the model's current state. Additionally, a left-to-right HMM is often used for speech models, 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.

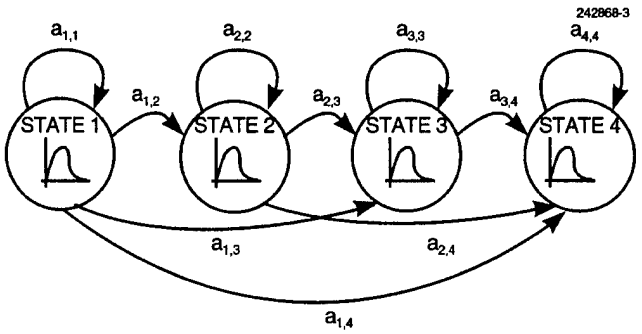


Figure 3. Example of a left-to-right hidden Markov model with four states.

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 trained on actual observation sequences from several spoken instances of "book" (see Figure 4). As before, each state represents the 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 the beginnings and endings of the different sounds have not been specified. Figure 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.

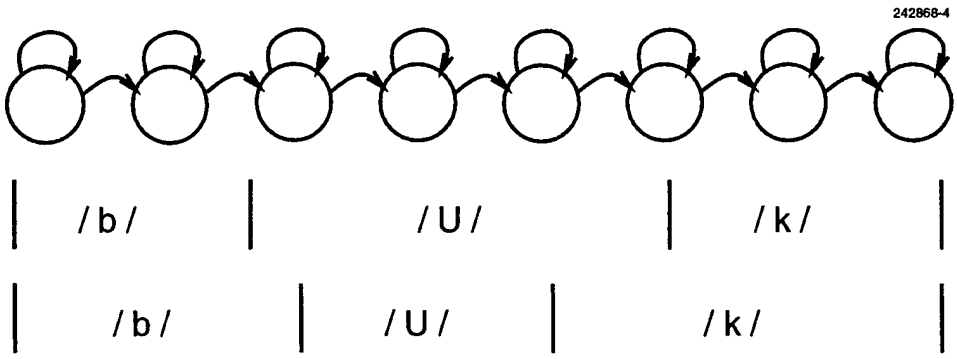


Figure 4. Hidden Markov model of "book," two possible transcriptions.

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 (1):

$$P(M|O) = P(O|M) * \frac{P(M)}{P(O)} \quad . \tag{1}$$

$P(M)$ 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. Because all the $P(M|O)$ calculations are compared with each other for the final decision, the $P(O)$ term can be thrown out, as its effects on each model probability are the same. $P(O|M)$ is calculated by recursively computing the likelihood that the model is in state j at time t, as in Equation (2):

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{i,j} b_j(o_t) \quad . \tag{2}$$

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; $b_j(o_t)$ is the probability that o_t was produced by state j; and N is the number of states. 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 each $\beta_t(i)$. 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 (3) [6]:

$$P(O|M) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i) \quad . \tag{3}$$

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

Subword Modeling and Phones in Context Often, 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 5. A word is recognized when its phoneme models lie, in proper order, on the most likely path through the network. Using this relationship with the "book" example, the HMMs of the system would model the /b/, /U/, and /k/ sounds instead of the word "book." A recognizer network including "book" would be built by connecting phoneme models for /b/, /U/, and /k/.

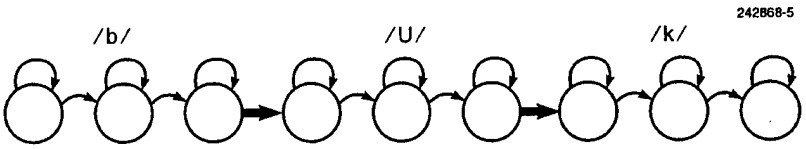


Figure 5. Example of "book" modeled with subword models.

One problem with straight phone-based models is that phones differ, depending on the sounds that precede or follow them, so that the HMMs actually used to model important words in recognition systems are often phones in context. For the "book" example, the actual HMMs would not be `b`, U, and `k`. They would be `#-b+U`, `b-U+k`, `U-k+#`, where `#` 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.

2.2 Hidden Markov Model Toolkit: An Overview

HTK [2] is a set of software tools designed to create, train, and test an HMM speech recognition system. In addition to the software tools, it includes standards to define, among other things, an HMM and an HMM network.

Creating an HMM system begins by defining the HMM models themselves. For the current system and task, described in Section 2.2.1, the word "card" was particularly interesting because it occurred so frequently and because the system often confused other words for "card." To model "card," four triphone models are necessary: `#-k+aa`, `k-aa+r`, `aa-r+d`, and `r-d+#`, where `#` denotes a word boundary. In the network definition, there is a mapping of "card" to those four triphones in succession.

Once all the HMM models are defined, programs within HTK are used to train the models. As this report does not focus on the training, it is sufficient to state that tools do exist that take as input a transcribed speech corpus and produce as output a trained set of HMMs for use with the HTK recognizer.

After a new HMM system has been trained, it is used in a recognition task. The main program used to recognize speech is `HVite`, which was the central focus of a major data analysis effort in this report (see Section 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 peak-picking mode was added at Lincoln Laboratory by supplementing the original `HVite` program. This peak-picking system was the subject of study in this report and is described in Section 2.2.2.

2.3 Word Spotting

2.3.1 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 [7]. The specific task addressed is that of the "credit card" based on the Switchboard telephone speech corpus [8]. In this task, the system looks for credit card-type words in a conversation about credit cards. The system does not need to transcribe the entire utterance. It merely needs to find where specific words were spoken. Figure 6 shows a segment of a switchboard conversation in which the word "card" was spoken.

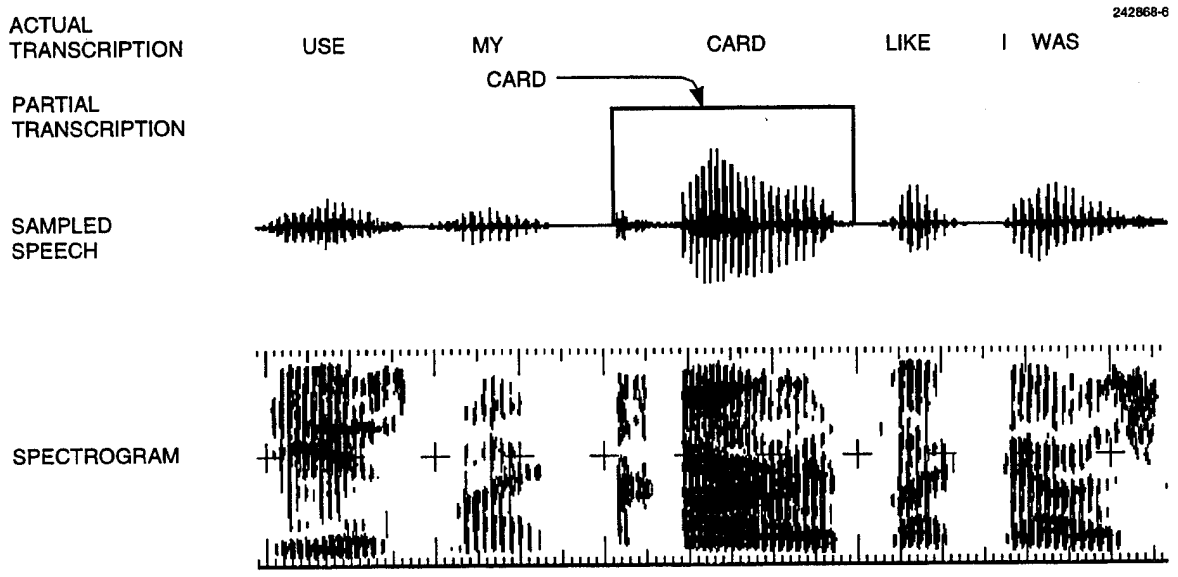


Figure 6. Example of word spotting for "card."

2.3.2 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 versus time, the probability usually peaks when the word actually ends. 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 ends of the words.

One issue in peak picking is the problem of normalizing the scores of the HMM output. Because 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 with a threshold value to determine if they should be labeled as putative hits (see Figure 7). Different segments of speech score differently, depending on factors such as speaker and communication channel, so normalizing the score becomes quite important. This system implemented a technique that normalizes the score of model frames $b_j(o_t)$ from Section 2.1.1 by subtracting a function of all the state probabilities as follows:

$$b_j'(o_t) = b_j(o_t) - f(b_1(ot), b_2(ot), ..., b_N(ot)) \quad . \tag{4}$$

Several different forms of the function $f()$ are currently being explored. Normalizing relative to the probabilities of all 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.

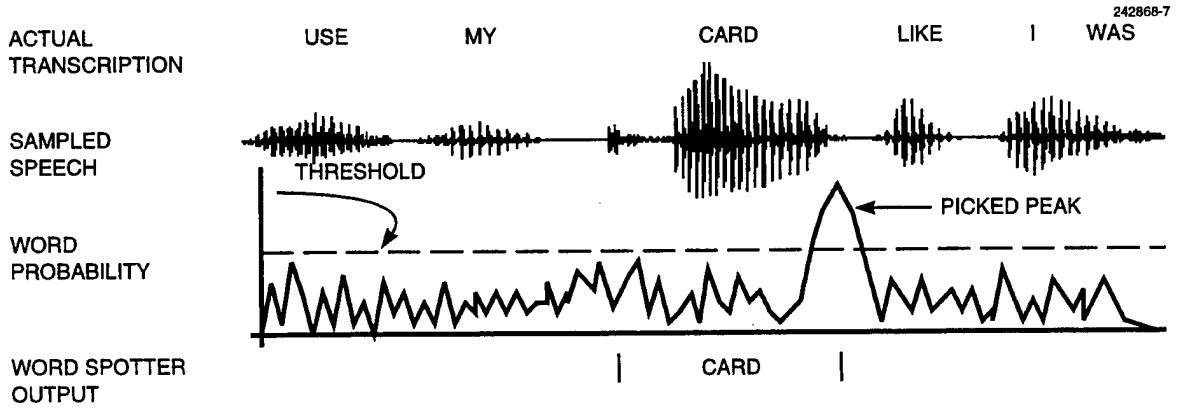


Figure 7. Example of peak picking for "card."

After these scores are calculated, the figure of merit (FOM) for the word spotter is calculated to measure overall system performance. The FOM is based on how many true instances of the word are 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.

$$FOM = \int_0^{10} P_d(FA) dFA \quad . \tag{5}$$

In Equation (5), 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.

2.3.3 Choice of the Word "card"

As mentioned in the preceding discussion, most of the data analysis work has been done for the word "card." One specific word was chosen because in the scope of this work it was more important to develop the data analysis techniques than to improve word spotter performance, although the two issues are closely related, as stated in Section 1.2.

Two main reasons for choosing "card" over the other 19 words in the credit card task are that 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. Second, "card" is a poorly scoring word. Of the frequently occurring words, "card" is the one on which the word spotter used in this study performs worst, i.e., has the lowest FOM. It is much easier to fix something clearly broken than something that works reasonably well.

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, one hopes to gain insights about where the system works and where it breaks down.

Figure 8 is an example of a best path calculation. Fifteen frames of data are being run against the model developed for "card." The system calculates several paths, which are pruned 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.

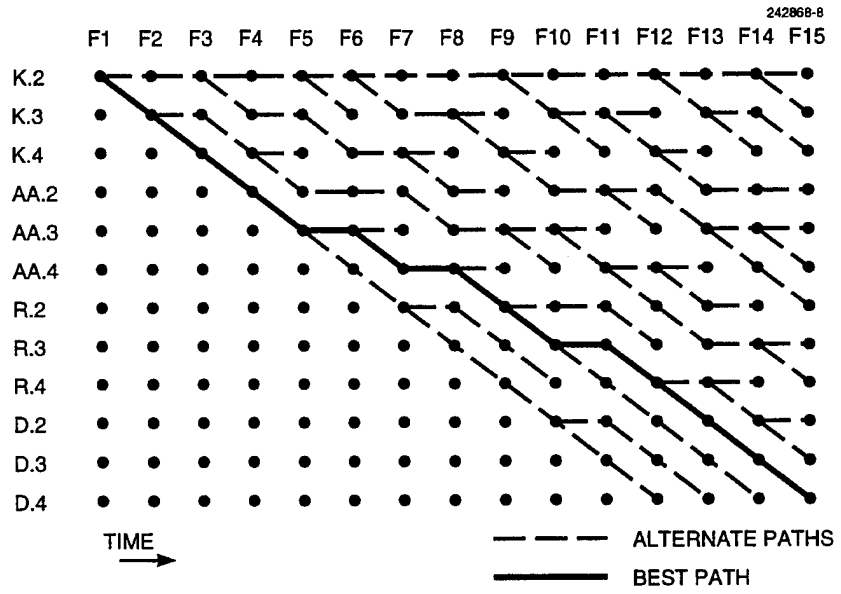


Figure 8. Example of backtrace decoding for "card."

3.2 Implementation

The two phases of producing a backtrace are gathering the relevant data and displaying the data in a meaningful way. Each phase is described next.

3.2.1 Data Gathering

To draw a complete backtrace, several pieces of information are needed. First, the per state location and score along the best path through the system need to be output at the end of a run. To provide the pictures with some context, the following additional pieces of important, but optional, information are useful:

- truth: transcript of what was really spoken,
- type: reference of whether a speech token was a hit or a false alarm, and
- filename: name of specific conversation from which this piece of speech came.

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 determine 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, therefore, was developed and added to the decoder to keep track of this information.

The current HTK systems use two methods of calculating when words occur. 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. In reference to Figure 8, the information along the solid line is output to a file for later use by the backtrace viewing program. In the second method, *peak picking*, several different word models are 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. Again, using Figure 8, this result could be either information from the best path along solid lines or from temporary paths along the dashed lines.

Because the HTK decoder program did not keep track of the backtrace down to individual frames of data, a specific backtrace data collection method was developed. A large linked tree that resembles the backtrace in Figure 8 was created; the actual structure is shown in Figure 9. Each node in the tree keeps track of the current frame, the model information (i.e., the 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., every $b_j(o_t)$ from Equation (4)]. To make this tree, each node needs a pointer to the node of the previous time from which came the best path to the current node in the current time. 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 calculates all the information needed for the backtrace. The new code needs only 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 (some with as many as 50,000 frames of data), a garbage collection scheme was developed. Without garbage collection, the program would have

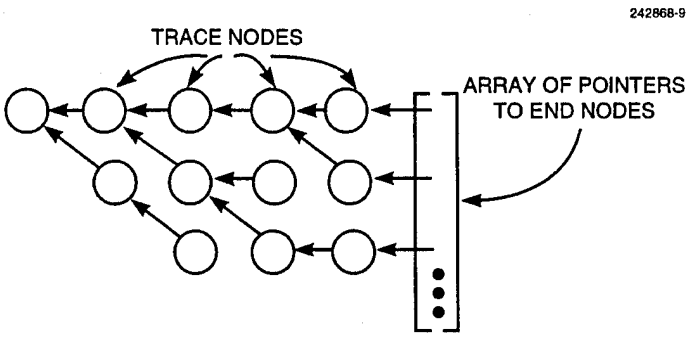


Figure 9. Picture of trace tree.

used on the order of 10^9 bytes of memory. Thus a field was added in each node to keep track of each successor. As successors to nodes were added, the number-of-successors field was incremented; as successors were freed, the field was decremented. When a branch of the tree had no successors, it was pruned, and its predecessor was checked to see if it also could be pruned. For example, referring to Figure 8, after 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. The result is shown in Figure 10. By using this method, only important branches of the tree would be tracked, resulting in reducing tree memory to below the memory required of the rest of the program.

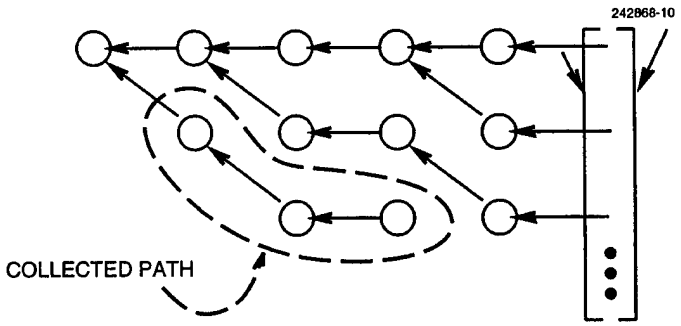


Figure 10. Garbage collection in the backtrace output code.

Two methods of output exist—one for the peak-picking decoder and one for the Viterbi decoder. The peak-picking decoder outputs trace information for each individual word as it is picked, then writes it to a file with the `.pptrace` extension. The Viterbi decoder waits until the

entire best path is calculated. Only at this time can decisions about the words be made, and then the backtrace can be output as well. This full backtrace information is written to a file with a `.trace` extension.

Other Backtrace Information. While the backtrace contains much information, other sources are needed to classify completely examples of the backtrace, giving the backtrace pictures a context in which they can be interpreted. Methods to obtain these additional pieces of information are detailed next.

The first useful piece of (external) information is "truth," which refers to what was actually said, in contrast to hypothesis, which refers to what the recognition system thinks was said. Files containing truth markings in an ASCII format are translated into a standard HTK format so that the system uses 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 study.

3.2.2 `vtrace` Viewer

Once all the necessary information is calculated and written to files on the system, the `vtrace` viewer draws its analysis pictures based on these files. Using the Motif library, `vtrace` was written in C. The original tool provided an initial GUI as well as code to read, sort, select, and view the individual trace segments. An enhanced tool improved and expanded upon the original tool's capabilities.

The user begins by selecting the trace file to view, as shown in Figure 11. Next, `vtrace` creates and presents an ordered list of all putative hits as shown in Figure 12. `vtrace` 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. The user clicks on the putative hit of interest, and `vtrace` displays it on the screen, as shown in Figure 13. Only the best path through the trellis is displayed 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 3 states each, and often span 80 or more frames of data. Displaying the data as a full trellis of that size would make visualization difficult.

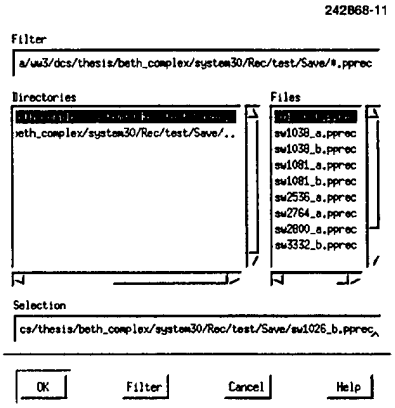


Figure 11. The **vtrace** *file loading dialog box.*

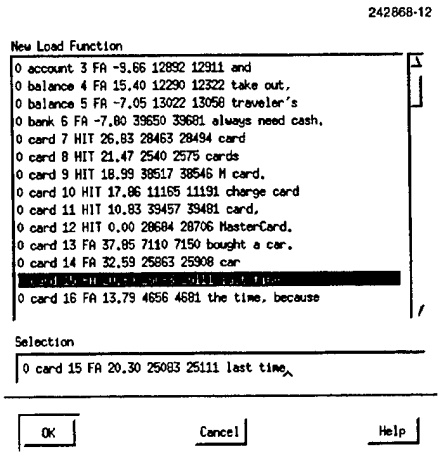


Figure 12. Putative hit selection dialog.

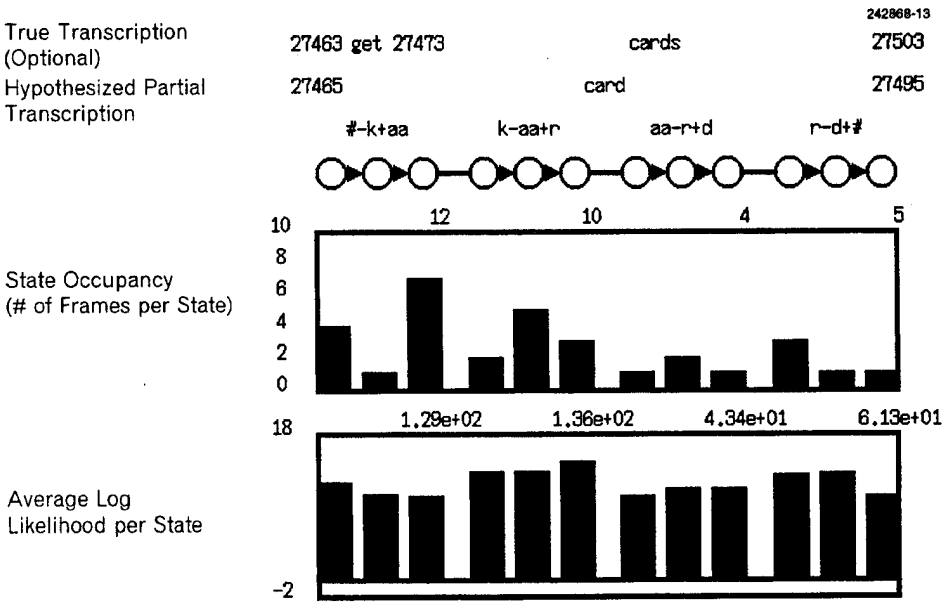


Figure 13. Output window for **vtrace** *with example putative hit.*

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 frequent false alarm for the "card" model: "car." Compare the true hit in Figure 14 with the false alarm in Figure 15. 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, making the overall word score lower than the true hit scores. However, the **r-d+#** model did score the end of "car" reasonably well, suggesting that the **r-d+#** model does not discriminate as well as it should.

For other putative hits, **vtrace** showed strange patterns of state occupancies (see Figure 16). The speech frames did not match the **#-k+aa** model well, which did not affect the overall score because the corresponding duration is much shorter than that of **#-k+aa**, as seen in Figure 14. The short duration prevented the expected accumulation of bad scores that should accompany segments of speech that match the model poorly. Because the rest of the utterance matched the rest of the model well, the utterance received a high score. In this way, abnormal occupancy patterns can allow a false alarm to score well. These kinds of observations led to the idea that improving the

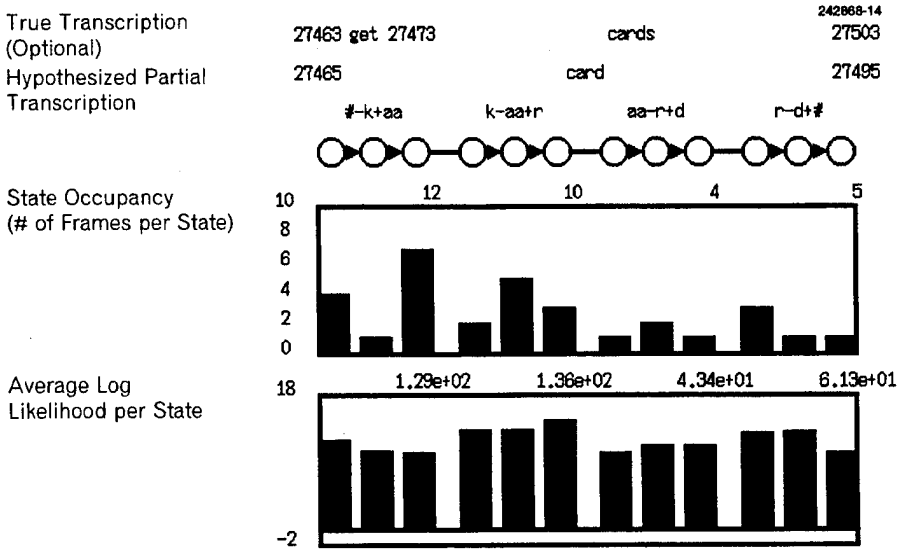


Figure 14. True hit for "card," score = 23.66.

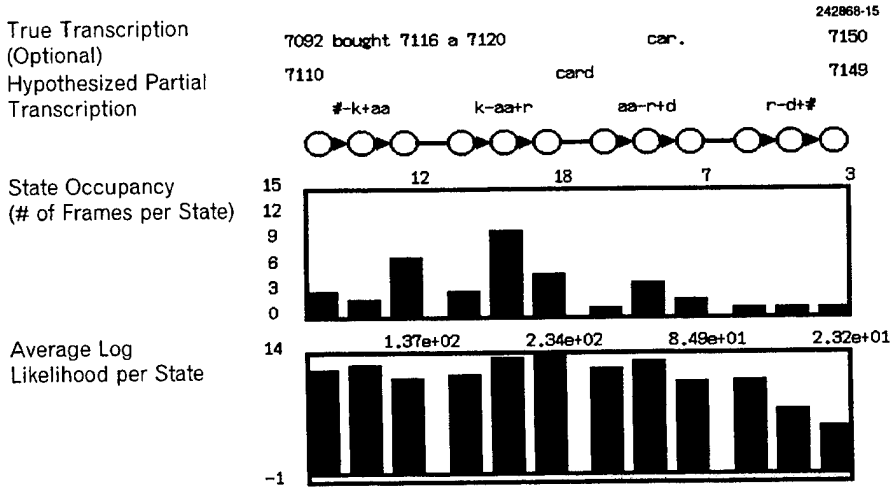


Figure 15. False alarm "car" for "card" model, score = 37.85.

modeling of durations in the HMMs might help improve word spotting performance. However, developing such a test was beyond the scope of this study as it involves significant additions to both the training and testing software.

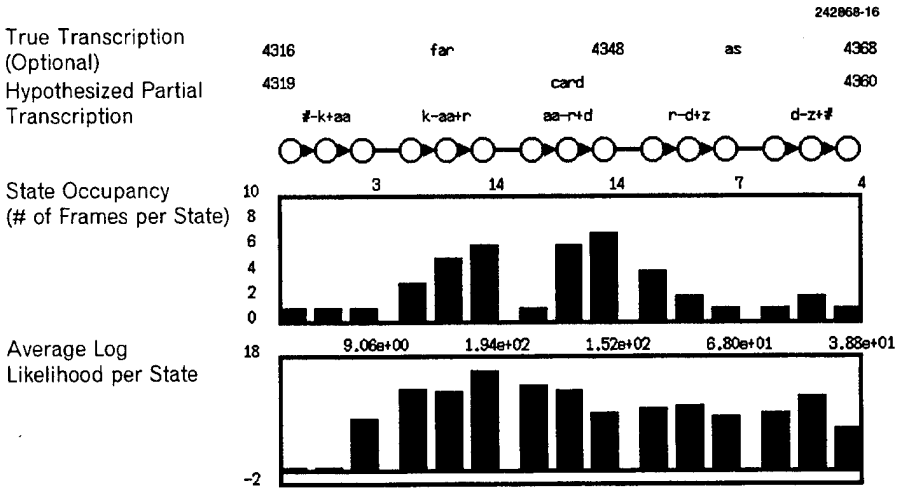


Figure 16. False alarm "far as" for "card" model, score = 25.07.

A second improvement idea for the word spotter was to limit the contribution that 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 them 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.

3.4 Secondary Test

After looking at several trace pictures such as those shown in Figures 14 through 16, 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 [9]. Because software existed to facilitate this type of test, such an experiment was run.

A multilayer 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 17. The training false alarms included all false alarms that scored higher than the lowest scoring true hit. The MLP then processed 140 putative hits from a set of testing data. The MLP used the putative hit per state occupancies and per state normalized log likelihood scores to remove from the putative hit list those entries judged to be false alarms. The secondary test did improve the performance of the system on the word "card" to 48.5 from the original FOM of 42.2.

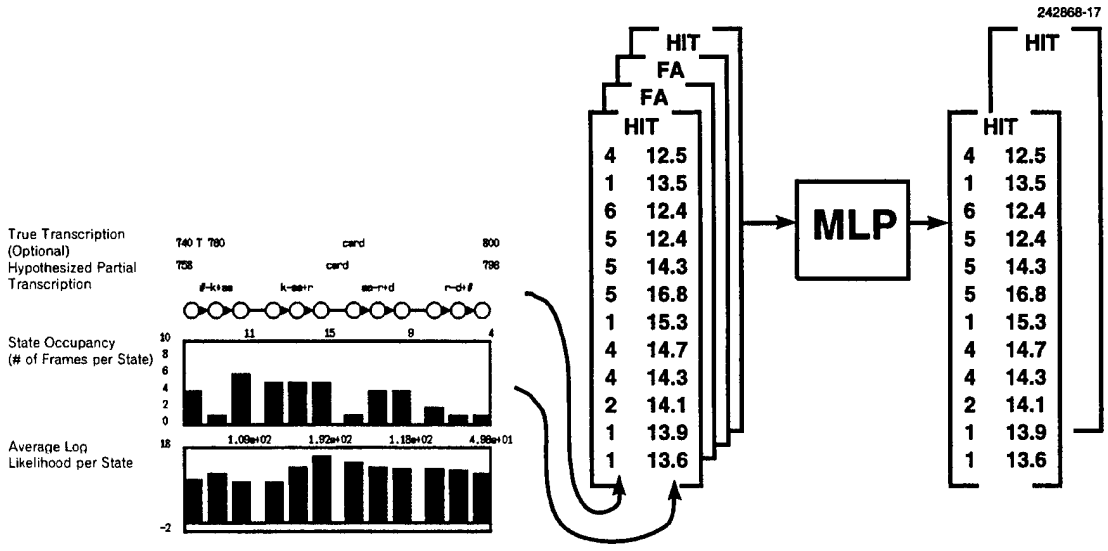


Figure 17. MLP classification on per state occupancies and per state normalized log likelihood.

This test was run again for words other than "card." Unfortunately, to date no similar results have been achieved, confirming previous experiences with secondary testing—sometimes it helps and sometimes it has no effect [11]. Because of this information, and because this line of research was off the principal subject of the report, no major effort was put into developing this test.

3.5 Conclusion

The backtrace analysis is useful because it provides a way to make sure the system is decoding as expected. Also, it has suggested several ways to improve the current word spotter. Perhaps these ideas will lead to improvement in system performance. Additionally, the work with the `vtrace` viewer program inspired a secondary test that yielded some promising results. Unfortunately, a number of the improvement ideas mentioned in Section 3.3 were too time consuming to fit into the scope of this project.

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 match the models. In the process of improving such methods, it can be useful to provide pictures that show the positions of one model relative to another or of a set of models relative to speech segments. If speech were represented using two parameters, it would be possible to display speech segments as well as models on a two-dimensional graph by assigning one parameter to the X axis and the other to the Y axis. However, in the system used here, speech is represented using two data streams: 12-dimensional cepstra and 13-dimensional delta cepstra. By using multidimensional scaling (MDS), data dimensionality can be compressed and meaningful two-dimensional plots can be produced. Unlike `vtrace`, the MDS plots are not exact representations of the data, but it is hoped that much of the meaning of the data can be retained in the scaling process.

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

4.2 Method

MDS provides a mechanism for projecting vectors in an N-dimensional space to a representation in lower dimension while attempting to maintain relative distances between vectors [12]. As seen in Figure 18, a three-step process converts raw data to an MDS plot. First, the distances between all the data points in the N-dimensional space are calculated and stored in a matrix. Second, the MDS program produces a two-dimensional representation of points related by the distance matrix. Third, the picture is plotted.

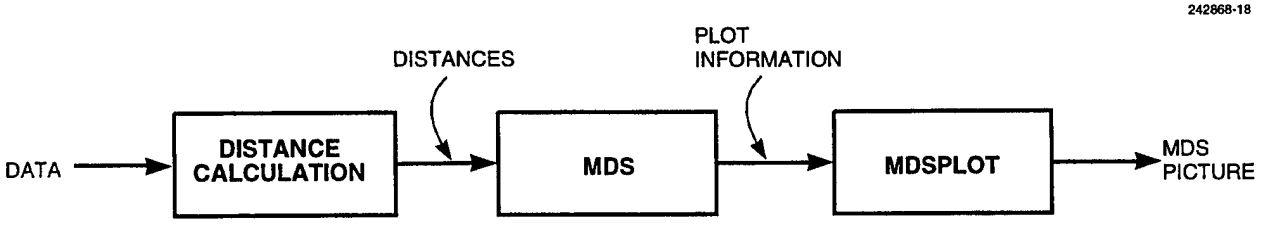


Figure 18. Three-step process for MDS.

4.2.1 Distance Calculation

The first part of the method shown in Figure 18 is the distance calculation. Three important factors are taken into consideration when choosing a distance metric for MDS—the distance must be inversely proportional to the nearness of the data points, the same method must compute distances between all points in a single MDS picture, and the chosen distance metric should relate in a meaningful way to the system being studied.

The two distance formulas described next 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 [13].

Mahalanobis Distance for Gaussian Random Vectors. The first distance metric used for MDS was a Mahalanobis distance [14], which is the Euclidean distance where each feature is weighted by its inverse covariance. In the case of diagonal covariance, the distance between the two vectors X and Y with diagonal covariance Λ can be calculated as follows:

$$X = \begin{bmatrix} x_1 \\ . \\ x_N \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ . \\ y_N \end{bmatrix}, \Lambda = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3^2 & \dots & 0 \\ . & . & . & & . \\ 0 & 0 & 0 & \dots & \sigma_N^2 \end{bmatrix}$$

$$d(X,Y) = (X-Y)^T \Lambda^{-1} (X-Y) = \frac{1}{N} \sum \frac{(x_i - y_i)^2}{\sigma_i^2} \quad . \tag{6}$$

While the Mahalanobis distance is similar to the word spotter distance between a speech vector and a model center, it is not identical. 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:

$$d(C_i, X) = (C_i - X)^T \Lambda_{C_i}^{-1} (C_i - X) \quad [15]. \tag{7}$$

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 5 centers closest to the speech vector and averages their distances. The distance to the different state density functions is based on these 5 closest centers as shown:

$$d(\lambda, X) = \log[\sum_{i=1}^{5} w_i \exp(d(C_i, X))] \quad , \tag{8}$$

where λ is the specific TGMM density function, X is the speech vector, C_1 through C_5 are the 5 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 (8).

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. It has proven useful for other speech processing problems such as speaker identification using Gaussian mixture models [13].

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 (see Figure 19), the expected probability of misclassification is defined as

$$\epsilon = \int \min(P_1 p(X|\lambda_1), P_2 p(X|\lambda_2)) dX \quad . \tag{9}$$

Because the calculation in Equation (9) has no closed form for GMMs, it can be approximated with the Bhattacharyya bound that calculates an upper limit on ϵ. For two Gaussian densities, $\lambda_1 = [\mu_1, \Lambda_1]$ and $\lambda_2 = [\mu_2, \Lambda_2]$, the Bhattacharyya bound is calculated as follows:

$$\epsilon \leq \epsilon_b = \int \sqrt{\min(P_1 p(X|\lambda_1), P_2 p(X|\lambda_2))} dX = \sqrt{P_1 P_2} e^{-\rho} \quad , \tag{10}$$

where

$$\rho = \frac{1}{8}(\mu_2 - \mu_1)\Lambda^{-1}(\mu_2 - \mu_1)^T + \frac{1}{2}\log\frac{|\Lambda|}{|\Lambda_1||\Lambda_2|} \quad . \tag{11}$$

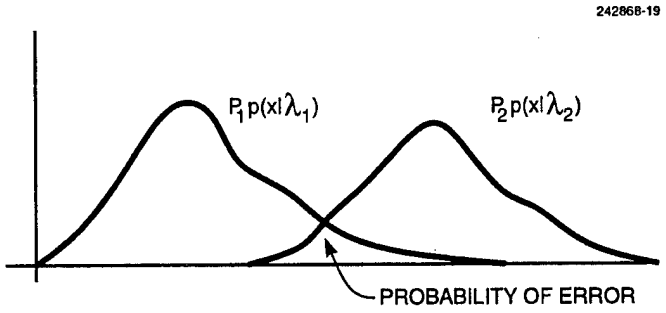


Figure 19. Depiction of the probability of error between two class distributions.

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

$$p(x|\lambda_1) = \sum_{i=1}^{M_1} p_i b_i(x) \quad , \tag{12}$$

$$p(x|\lambda_2) = \sum_{j=1}^{M_2} q_j a_j(x) \quad , \tag{13}$$

in which $b_i(x)$ and $a_j(x)$ are Gaussian densities of order M_1 and M_2, respectively. The distance used was

$$d(\lambda_1, \lambda_2) = -\log\left[\sum_{i=1}^{M_1} \max_{1 \le j \le M_2} \sqrt{p_i q_j}\exp -\rho(i,j)\right] \quad , \tag{14}$$

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

Beth Carlson (a researcher in Group 24 at Lincoln Laboratory) wrote the `HHDist` program to calculate this distance for HTK models. With a few minor changes, it produced distances that 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 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 affect performance. Therefore, the Bhattacharyya distance was employed to help gain intuition into the positioning of models.

4.2.2 Multidimensional Scaling Method

A basic overview of the MDS method [12] shows that it 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 vectors X_i and store the distances in the set of vectors X in a matrix Δ shown in Equation (15).

$$\Delta = \begin{bmatrix} \delta_{1,1} & \delta_{1,1} & \dots & \delta_{1,N} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,N} \\ \cdot & \cdot & & \cdot \\ \delta_{N,1} & \delta_{N,2} & \dots & \delta_{N,N} \end{bmatrix} \quad . \tag{15}$$

The distance between two points x_i and x_j would be calculated as in Equation (16), where δ is some meaningful distance function such as those described in Section 4.2.1:

$$\delta_{i,j} = \delta(x_i, x_j) \quad . \tag{16}$$

Once Δ has been computed, all vectors X_i in X are scaled using the following iterative process. First, they are placed in R dimensional space, where initially $R = M$, the number of dimensions in each vector X_i. Once the points have been placed, a dimension is removed, and the points are replaced in this lower dimensional space using a stress minimization process. 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 in a four-step process. First, $d_{i,j}$ is calculated for each i and j based on their initial position, as in Equation (17), where $x_{i,r}$ is the rth dimension of x_i:

$$d_{i,j} = \sqrt{\sum_{r=1}^{R}(x_{i,r} - x_{j,r})^2} \quad . \tag{17}$$

Next, the "**f-stress**" of the current X placement is calculated, as seen in Equation (18), where $f(\delta_{i,j})$ is the objective function that relates proximity δ to distance in the space d:

$$\textbf{f-stress}(\Delta, X, f) = \sqrt{\frac{\sum_i \sum_j [f(\delta_{i,j}) - d_{i,j}]^2}{\sum_i \sum_j d_{i,j}^2}} \quad . \tag{18}$$

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

$$\textbf{stress}(\Delta, X) = \min_{f}^{\arg} (\textbf{f-stress}(\Delta, X, f)) \quad . \tag{19}$$

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 (20):

$$\textbf{stress}(\Delta, \hat{X}) = \min_{X}^{\arg} (\textbf{stress}(\Delta, X, f)) \quad . \tag{20}$$

While the preceding process is useful to define the best fit, it does not provide a good way to calculate $\hat{X}$ given Δ. For this, Kruskal and Wish [12] 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 anywhere on this terrain and letting it find the lowest point 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 starting with $R = M$ 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. However, becoming trapped in a local minimum is rare [12]. (The current study used a 1973 version of the MDS program, which could only scale up to 62 points at a time. Newer programs do not have this limit, but they were not available for this work.)

4.2.3 Output: `mdsplot`

The output of the MDS program can be converted into standard X and Y coordinates, and simple plotting packages already exist to display the picture. However, to facilitate the display of speech models and speech segments, a special plotting program was created by Linda Sue Sohn (staff

member of Group 24 at Lincoln Laboratory)—`mdsplot` 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 that minimizes the difference between actual and displayed distances, the meaning of the two dimensions in the final picture is arbitrary 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 that scored best in the card model) was an instance of the word "car." In fact, for FOM 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 "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. The models were TGMMs with 128 model centers per model state; only the 2 or 3 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 20. The circles represent the different model centers and are included only for display purposes. The center labels describes each specific center as well as the model it represents. For example, the model label `d.35` refers to the 35th Gaussian center, which represents a center 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 the word from which the data came, as well as the specific frame 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 centers, suggesting that while both tokens of the /k/ sounds appear to be close

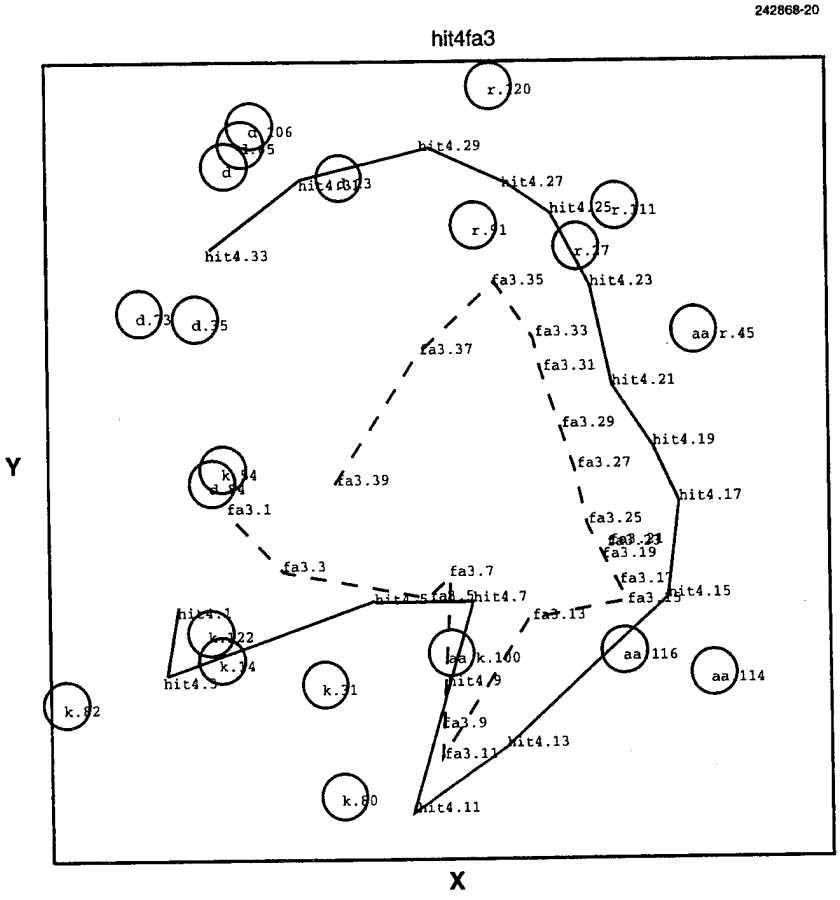


Figure 20. MDS output for Experiment 1: hit versus false alarm.

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 from the `r-d+#` model, suggesting that the `r-d+#` model did not affect scoring much and 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 (Figure 21), which can be compared with the hit versus false alarm picture (Figure 20). Figure 21 differs from the hit versus false alarm case as expected because the 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 Figures 20 and 21 is that the similar data points are not in similar locations. The word tracks in Figure 20 proceed counterclockwise while the tracks in Figure 21 proceed clockwise. The MDS procedure is not guaranteed to produce similar pictures for similar data, and caution must be used when interpreting MDS pictures.

Figure 22 shows one example where the tracks for "car" and "card" were very dissimilar, and the plot is a bit more difficult to interpret. However, the actual vector distances in Table 1 show 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, whereas `hit5.1` and `fa5.1` had a Mahalanobis distance of 5.6. These distances are reasonably reflected in the MDS picture.

TABLE 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

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 is used to create an MDS plot of the monophone set from the word spotter in the

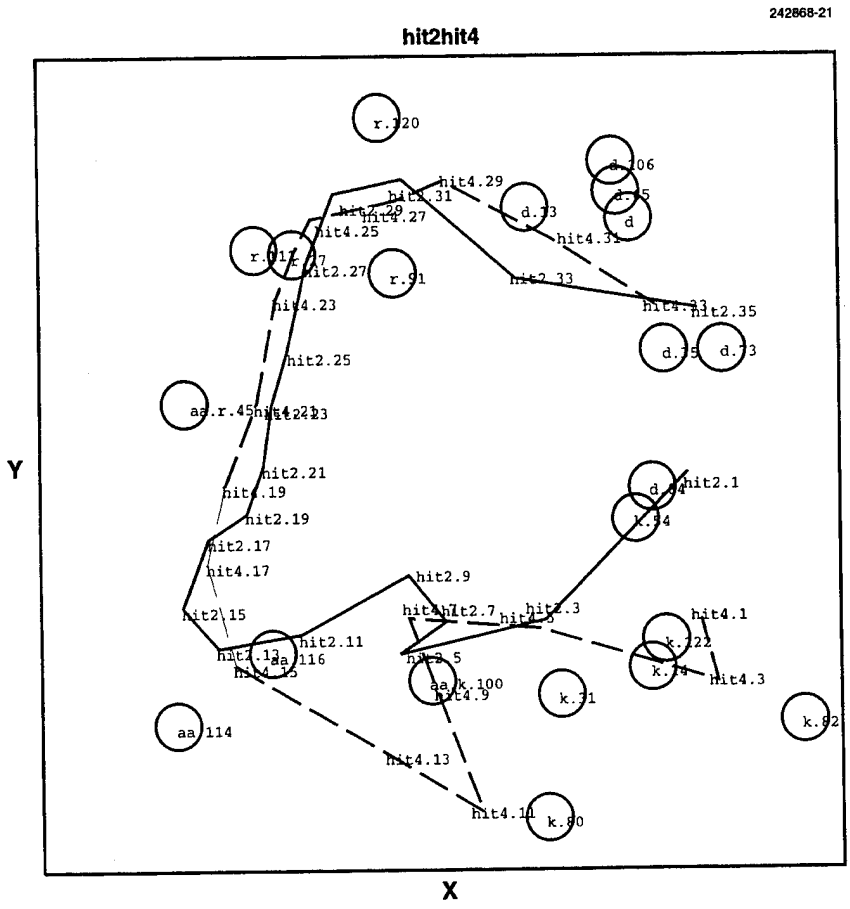


Figure 21. MDS output for Experiment 1: hit versus hit.

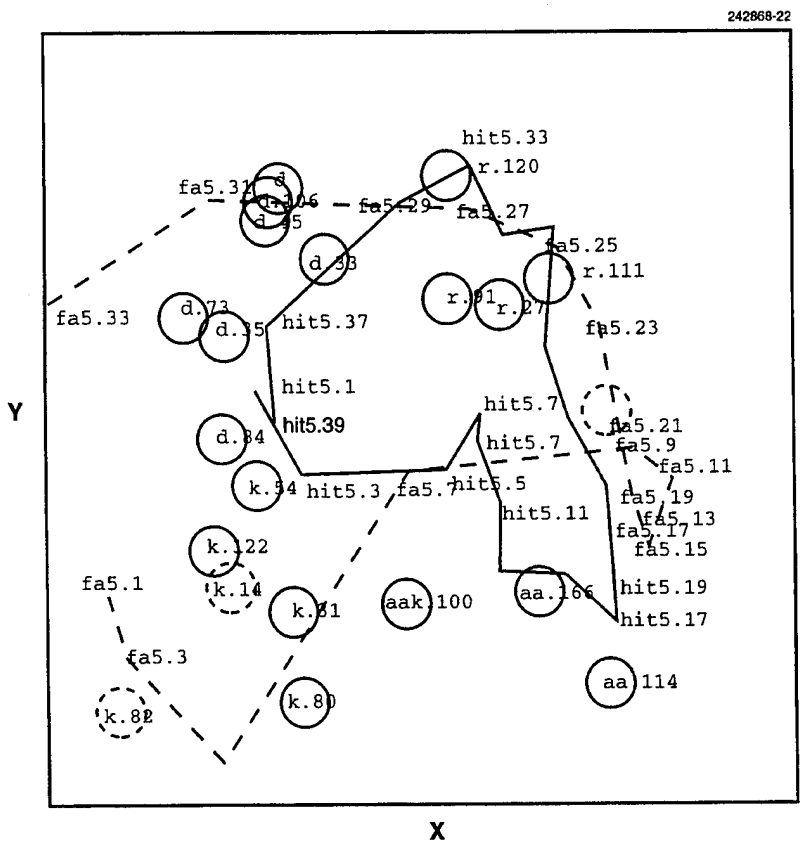


Figure 22. MDS output for Experiment 1: alternative hit versus false alarm.

hope that such a plot would provide insight as to the models that might cause confusion for the system.

Figure 23 shows that most of the models were clustered (as one would expect) by phonetic class, while a few seemed to be misplaced [16]. Some of the phones in unexpected locations include the **dx** and **k** models. Although both were expected near the other stops, the **dx** model is not. However, the Bhattacharyya distances to these models, shown in Table 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 2, do not support this placement 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, which the MDS picture reflects, showing that while MDS can assist in viewing data that are otherwise difficult to display, results must be checked against the originals.

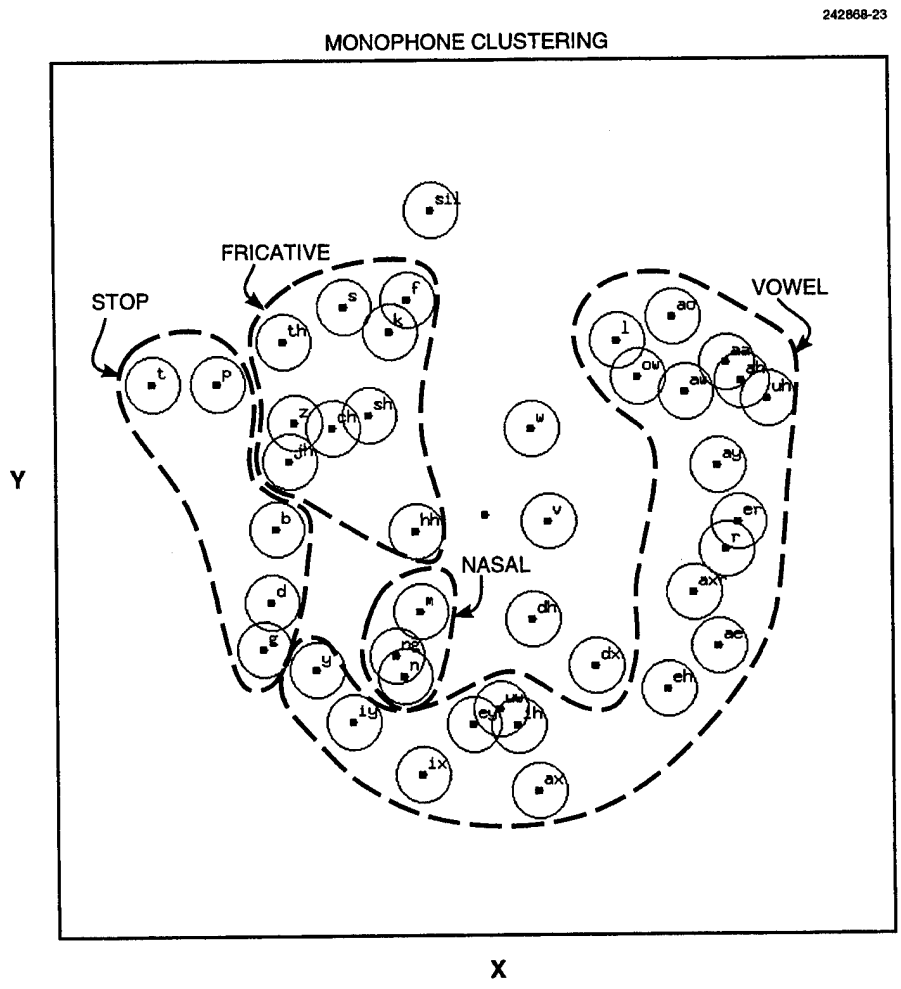


Figure 23. MDS output for Experiment 2.

TABLE 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

4.5 Experiment 3: Amounts of Training Data

A final experiment involved exploring the differences between models that had been trained on different amounts of the training data. Because Switchboard recognition runs had been made using models trained on various amounts of data, it was desirable to see the effect that 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 with 75%; set 3 with 50%; and set 4 with 25%.

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, all 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 are seen in Figure 24. The circles point out the actual model locations. 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 24 is shown in Figure 25. Three specific examples are traced—**uw**, **ng**, and **er**. Here it seems as though the models are exchanging positions relative to each other for each new data set (note the distances in Table 3). The distance between **ng.2** and **uw.4** is larger than the distance between **ng.2** and **ng**, but the MDS plot in Figure 24 places **ng.2** and **uw.4** close while it places **ng** farther away, showing that the scaling procedure can produce misleading pictures.

TABLE 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 second experiment involves displaying all training sets for all models. Because this experiment involves many data points, the models were divided into three groups, based on the experiment in Section 4.4. Figure 26 displays one of the three MDS plots, although similar results were seen in the other two pictures as well. Figure 27, which is easier to analyze, shows a subset of the information of Figure 26. While some models seem to cluster upon 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 decrease while the amount of training data increase. It also appears as though the **t** model is not converging on any point relative to the other models. However, the picture is not

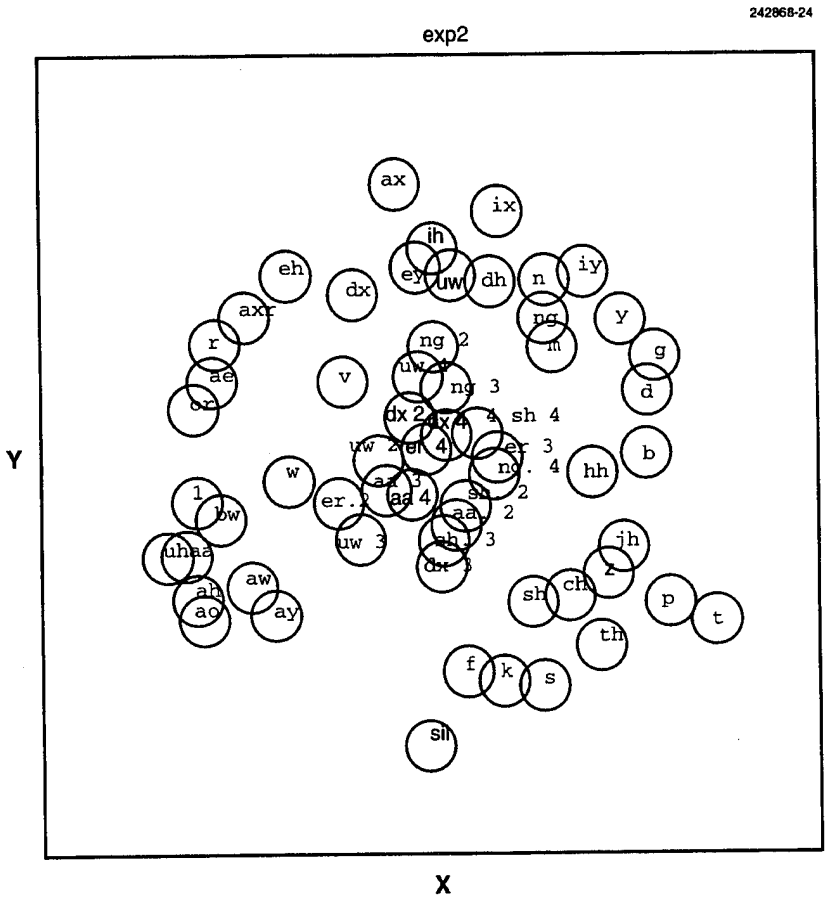


Figure 24. Output for MDS Experiment 3: all monophones.

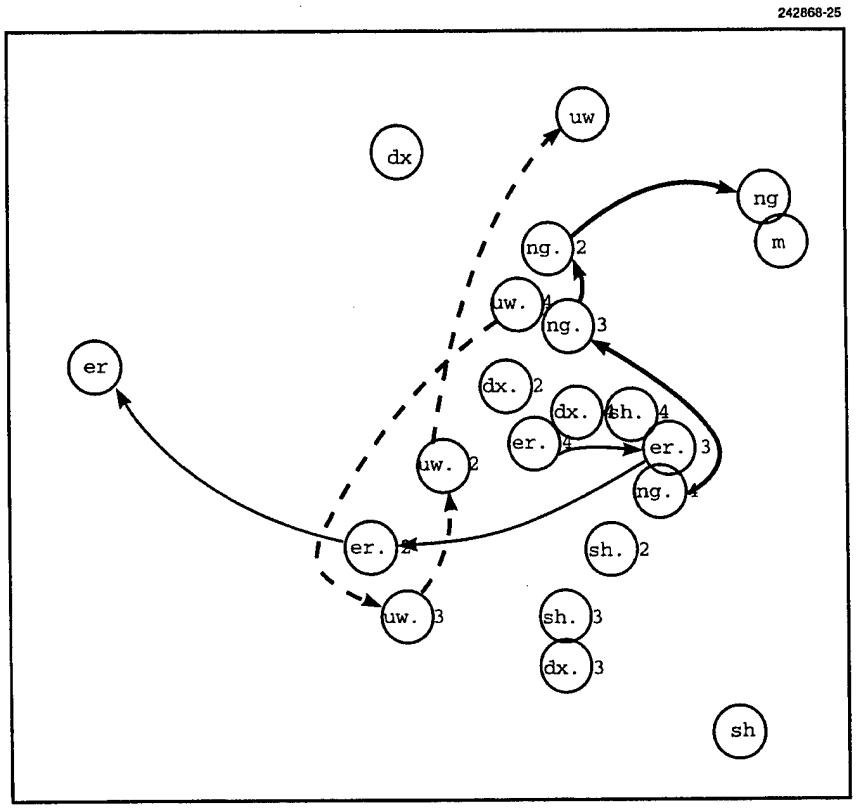


Figure 25. Partial output of Experiment 3: all monophones.

representative of the data shown in Table 4. The closest model to `k` is `t`, according to the data, whereas the 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 provides 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 difference could be seen using the MDS method. However, MDS plots that were ambiguous show that the trends must be confirmed with further analysis of the original data.

The second experiment demonstrated that this method allows visualization of groups of models that might be confused for each other. The MDS picture placed most monophones in a reasonable way relative to each other, but a few monophones seemed to be misplaced.

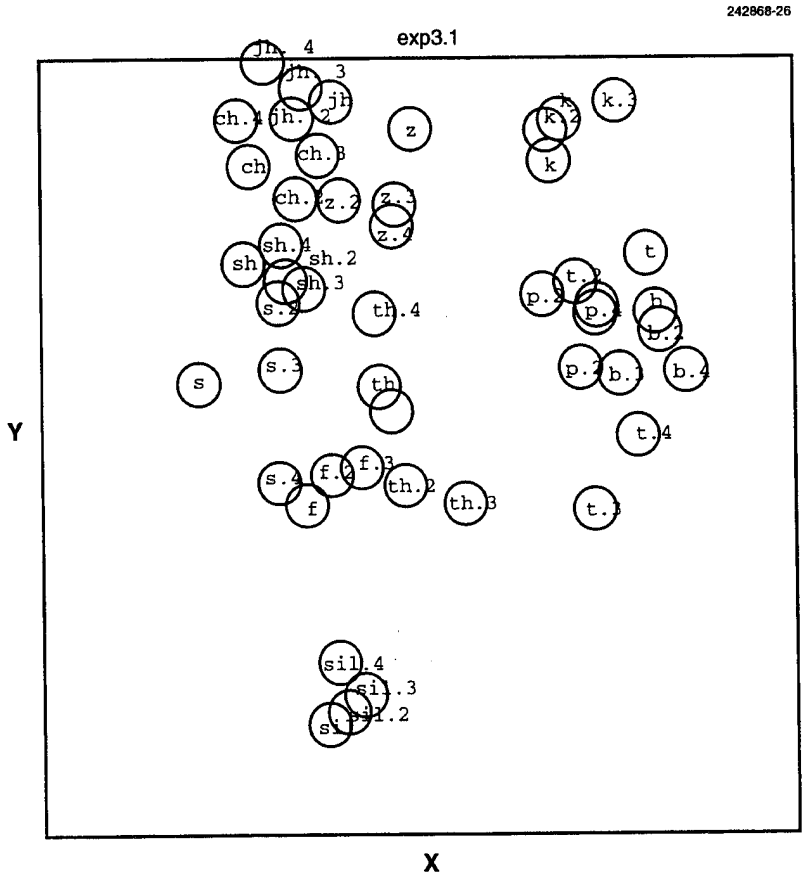


Figure 26. MDS Experiment 3: partial monophone list.

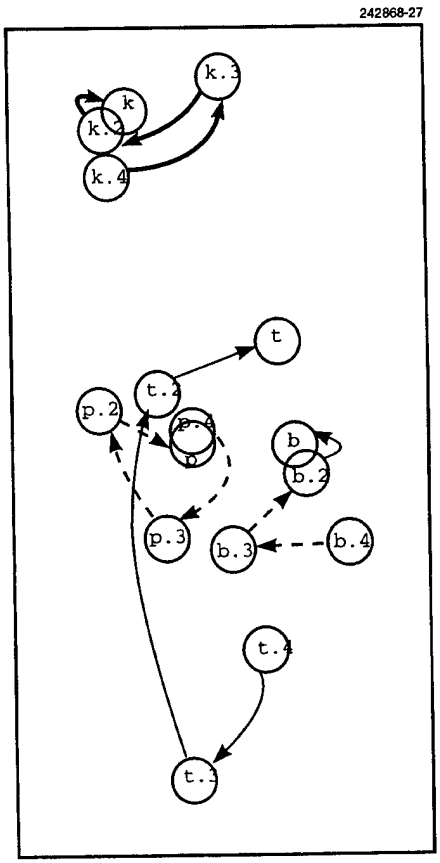


Figure 27. Partial output of Experiment 3: partial monophone list.

TABLE 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	–

The third experiment demonstrated the effects of training on the confusion 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 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 that had been scaled in a single invocation of MDS, and it would have been helpful to have added more for specific comparisons. New software handling 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 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 prescaled domain into the postscaled domain prevented such analysis.

Additionally, while the experiments in Sections 4.4 and 4.5 explore monophones, the word spotter uses triphones. Therefore, the same experiments should be conducted using triphones.

5. CONCLUSION AND FUTURE WORK

The exploratory data analysis provided some insights into an HMM word-spotting application. The backtrace viewer showed the effect of state occupancy on score, which led to a secondary test that improved word-spotting system FOM for at least one word. The MDS technique showed the progression of a word through the different models, the clustering of different model centers and models, and the location of models moving about (relative to each other) as they are trained. Some ideas for future work are discussed next.

5.1 Improvements to the HTK Word Spotter

First, adding a better duration model to HTK might improve word spotter performance. Data analysis suggests that odd paths through the models are 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 seems clear that some models do not capture the full complexity of the sounds they intend to model. From the MDS analysis, it seems that the models are not converging on a specific point, suggesting that the architecture of the models is insufficient. Perhaps a different model architecture would be more easily trained on the data.

Third, the MDS analysis also suggests that certain sounds are difficult to distinguish. Adding a system to the HTK word spotter that could notice a sound as possibly within an ambiguous group and then perform 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 the insights gained by using them suggests that this general field of study should be explored further. Additional data analysis techniques have 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 28 for an example). Recall that $b_j(o_t)$ is the probability that observation o_t was produced by state j. This method 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 to compare $b_j(o_t)$, the state probability scores, with o_t, the observations.

Referring to the "book" example, one would expect the probability that the speech was produced by the b state would be highest during the time the /b/ phoneme was produced, and

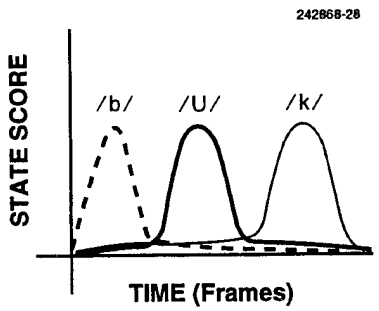


Figure 28. State score versus frame index.

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 the HMM, its output is determined by state-dependent probability density functions. To make useful probability density functions describing the speech observations, assumptions must be made about the distribution of the speech vectors (e.g., are they modeled accurately by Gaussian mixture densities). As demonstrated in Figure 29, these assumptions can be checked by plotting the assumed distribution of feature vectors against the actual distributions for both the training and the testing data. This comparison could ensure that the data for a specific state's model fits the overall probability density function implementation, and if this were not the case, the model's probability density function could be altered to reflect the poor fit. 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.

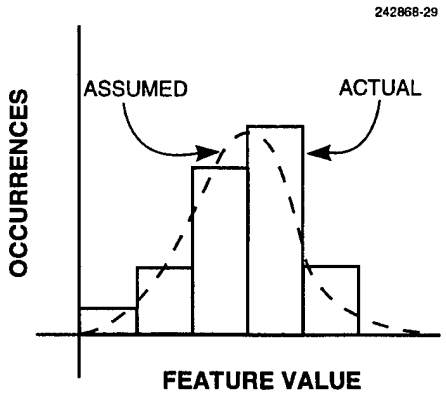


Figure 29. Real versus modeled feature distributions.

REFERENCES

1. L.R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE* **77**, 257–286 (1989).
2. S.J. Young and P.C. Woodland, *HTK Version 1.4*, Cambridge, England: Lynxvale, Ltd.
3. J.W. Tukey, *Exploratory Data Anlysis*, New York: Addison-Wesley Publishing Company (1977).
4. J.N. Marcus, *Word and Subword Modeling in a Segment-Based HMM Word Spotter Using a Data Analytic Approach*, Ph.D. thesis, Massachusetts Institute of Technology (September 1992).
5. A.W. Drake, *Fundamentals of Applied Probability Theory*, New York: McGraw-Hill Book Company (1967).
6. D.B. Paul, "Speech recognition using hidden Markov models," *Linc. Lab. J.* **3**, 41–62 (1990).
7. J.R. Rohlick, W. Russell, S. Roukos, and H. Gish, "Continuous hidden Markov modeling for speaker-independent word spotting," *ICASSP '89 Proc.* **2**, 627–630 (1989).
8. J.J. Godfrey, E.C. Holliman, and J. McDaniel, "Switchboard: Telephone speech corpus for research and development," *ICASSP '92 Proc.* **2**, 517–520 (1992).
9. J.R. Rohlick, P. Jeanreaud, K. Kg, H. Gish, B. Musicus, and M. Siu, "Phonetic training and language modeling for word spotting," *ICASSP '89 Proc.* **2**, 459–462 (1989).
10. R.P. Lippmann, L. Kukolich, and E. Singer, "LNKnet: Neural network, machine-learning and statistical software for pattern classification," *Linc. Lab. J.* **6**, 249–268 (1993).
11. R.P. Lippmann and E. Singer, "Hybrid neural-network HMM approaches to wordspotting," *ICASSP '93 Proc.* **1**, 565–568 (1993).
12. J.B. Kruskal and M.Wish, *Quantitative Applications in the Social Sciences* **7-011**—*Multidimensional Scaling*, Newbury Park, Calif.: Sage Publications (1978).
13. D. Reynolds, private communication (April 1993).
14. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, New York: John Wiley and Sons (1973).
15. B.A. Carlson, *A Projection-Based Measure for Automatic Speech Recognition in Noise*, Ph.D. thesis, Georgia Institute of Technology (November 1991).
16. D. O'Shaughnessy, *Speech Communication, Human and Machine*, New York: Addison-Wesley Publishing Company (1987).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	25 October 1995	Technical Report

4. TITLE AND SUBTITLE
Graphical Analysis of Hidden Markov Model Speech Recognition Experiments

5. FUNDING NUMBERS
C — F19628-95-C-0002
PR — 279-2-202

6. AUTHOR(S)
Dewitt C. Seward IV and Marc A. Zissman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lincoln Laboratory, MIT
244 Wood Street
Lexington, MA 02173-9108

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-1009

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Defense
Washington, DC 20301-7100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-94-110

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

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 multidimensional scaling display. These tools were built using the HMM Toolkit. Use of the Viterbi backtrace tool provided insight that eventually led to improved recognition performance.

14. SUBJECT TERMS
speech recognition
wordspotting
multidimensional scaling
hidden Markov models
exploratory data analysis

15. NUMBER OF PAGES
60

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Same as Report