Automatic Phoneme Recognition with Segmental Hidden Markov Models

Areg G. Baghdasaryan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

January 27, 2010
Blacksburg, Virginia

Keywords: Speech, Speaker, Viterbi, Baum Welch, Hidden Markov Model, Segmental HMM, Cluster

Automatic Phoneme Recognition With Segmental Hidden Markov Models

Areg G. Baghdasaryan

(ABSTRACT)

A speaker independent continuous speech phoneme recognition and segmentation system is presented. We discuss the training and recognition phases of the phoneme recognition system as well as a detailed description of the integrated elements. The Hidden Markov Model (HMM) based phoneme models are trained using the Baum-Welch re-estimation procedure. Recognition and segmentation of the phonemes in the continuous speech is performed by a Segmental Viterbi Search on a Segmental Ergodic HMM for the phoneme states.

We describe in detail the three phases of the phoneme joint recognition and segmentation system. First, the extraction of the Mel-Frequency Cepstral Coefficients (MFCC) and the corresponding Delta and Delta Log Power coefficients is described. Second, we describe the operation of the Baum-Welch re-estimation procedure for the training of the phoneme HMM models, including the K-Means and the Expectation-Maximization (EM) clustering algorithms used for the initialization of the Baum-Welch algorithm. Additionally, we describe the structural framework of - and the recognition procedure for - the ergodic Segmental HMM for the phoneme segmentation and recognition. We include test and simulation results for each of the individual systems integrated into the phoneme recognition system and finally for the phoneme recognition/segmentation system as a whole.

Acknowledgments

I would like to thank my parents for showing me the importance of graduate education and their continuous trust and support towards successful completion of my thesis work. Their genuine support helped tremendously during my graduate career.

Also, I would like to thank my graduate advisor Dr. A. A. (Louis) Beex, for giving me the opportunity of working as part of the team in the Digital Signal Processing Laboratory (DSPRL) at Virginia Tech. I'm also grateful to him for giving me the chance to work on advanced projects that extended my experience, and helped me pursue my education. Under his supervision, my work in the DSPRL has been productive and inspiring. I deeply appreciate his insightful help and support for the completion of this thesis work.

I thank Dr. Chris L. Wyatt and Dr. Claudio da Silva for their time in reviewing this thesis work. Their comments are greatly appreciated. I also give thanks to my lab mates Roshin Pally and Amy Malady. Special thanks to my roommate William Chan for making my graduate school experience fun and productive. Also special thanks to my sister Ani Baghdasaryan for cheering me up in difficult times.

This work received support from the DSPRL at Virginia Tech.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Communication is a vital part of everyday human life. Particularly, voice communication, since it is the primary method of communication between humans and is defined by passing verbal information from the speaker to the listener. There are other means of communication between people, however the flow of information via speech is considered to be the fastest and the most effective. Thus it is crucial to be able to obtain as much information as possible from a speech signal. Due to the limited training data available, most of the large vocabulary Automatic Speech Recognition (ASR) systems use several interconnected layers of recognition for optimum performance. Phonemes are perhaps the most common sub-word modules that are used in ASR systems. Thus, in this thesis we concentrate on the task of phoneme recognition, which plays a key role in the recognition of continuous speech using ASR systems.

1.1 Background on Speech Processing

The speech generated and perceived by people is a set of periodic variations of pressure propagating through the air. These air vibrations are fueled by the human lungs. The vocal tract subsystem of the human auditory system is responsible for the shaping and the production of the actual sound. The human vocal tract consists of the pharynx, mouth, and nose cavities. The air generated by the lungs goes to the human glottis. This generated air is responsible for the production of the vowels and voiced sounds and also generates a pulse train for the vocal tract. A noise generated by the human glottis results in consonants or unvoiced sounds [1].

Initially the variations in the air pressure are converted into an analog signal by the use of a microphone or a telephone headset. Then the analog signal is converted into a digital signal by the use of the analog to digital converter. An anti-aliasing low-pass filter is placed directly before the digital sampled speech for the rejection of any would-be high-frequency aliasing components, channel noise, and/or inter-channel interference. Ideally, the low-pass anti-aliasing filter would have a cut-off frequency at half of the Nyquist frequency, which corresponds to half of the sampling frequency.

Speech recognition refers to the extraction of verbal information from the speech utterance. In other words, a speech recognition system takes the speech utterance as the input and produces a text output that corresponds to the given speech. History of speech and speaker recognition dates back to the 1870s when Alexander Graham Bell was experimenting with speech in an effort to make human speech visible for people with hearing disabilities. He was the first to experiment on how to convert the variations in the air pressure generated by the human voice into electric signals. The development of speaker identification systems started in the 1960s when scientists found that the characteristics of the human voice can be used to uniquely identify the speaker. Speech recognition systems are widely used in the health care industry, in military applications such as battle management and air traffic control, and for telephone applications [2]. The performance of speech recognition systems depends on several factors, such as the vocabulary size, the amount of training, and the computational complexity of the system.

Automatic speech recognition systems have several unique and defining characteristics, and are categorized into several different types. The speech recognition system can be either a speaker independent or a speaker dependent system. The speaker dependent system is able to reliably extract verbal information from a particular speaker, whose acoustic characteristics are recorded in a speech database. On the other hand, a speaker independent recognition system is able to extract the verbal information from any speaker (whose acoustic data may not be installed in the database). Speaker independent systems are computationally more complex than speaker dependent systems, because of the need to recognize speech produced by a speaker that is not prerecorded [3]. The recognition systems developed for this thesis are speaker independent.

An Automatic Speech Recognition system can also be categorized as an isolated-word recognition system or a continuous speech recognition system. In an isolated word recognition system, there is a distinct pause - of about 200 ms - between the words in the spoken utterance, thus a simple endpoint detection algorithm is sufficient for the segmentation of the words. Thus, the recognition task of the isolated word recognition system is reduced to an accurate segmentation of the words followed by single word recognition. On the other hand, a continuous speech recognition system is responsible for recognizing a sequence of words that are spoken without a pre-specified pause between the uttered words. Thus, in a continuous speech recognition system the visible set of pauses between the words are generated by the inter-word fricatives and gaps. The continuous speech recognition system is generally more computationally complex than the isolated word recognition system. Also, the performance of continuous speech recognition systems is generally degraded by the co-articulation effects between the words. This thesis focuses on the recognition of a string of phonemes in a continuous speech recognition system.

The speech recognition system can also be characterized by the size of the vocabulary supported. There are three general ranges of vocabulary sizes: small, medium, and large containing 1-99 words, 100-999, and more than 1000 words respectively. As the vocabulary size increases, so does the memory size and the computational complexity of the speech recognition system [3]. Both the computational complexity and the memory usage of the recognition system are directly proportional to the vocabulary size. Generally the large vocabulary automatic speech recognition systems use a layer of sub-word modules, such as phonemes, to compensate for the lack of available training data [4]. Thus the recognition efforts in this thesis concentrate on the task of phoneme recognition.

The earliest speech recognition systems used the information in the speech signal, such as the signal energy and the signal zero-crossing rate, in an attempt to recognize an isolated word utterance. Each of the words in the dictionary had a set of parameters, generated from the time domain speech signal. Then these parameters, of each word, were compared with the set of parameters from the spoken utterance. Research has shown that the parameter extraction in the time domain from the speech signal is not effective and does not produce high recognition rates, since the speech signal typically has a high variability. Thus the speech recognition task was classified under the pattern recognition task. Research in the 1970s has shown that modeling of the vocal chamber and the pitch of the speaker can produce a higher recognition rate. This was achieved using Linear Predictive Coding (LPC) [5]. Later, the cepstral representation of the power spectrum of the speech signal was shown to lead to good recognition results for the case of isolated word recognition. The Mel-Frequency Cepstral Coefficients (originated by Paul Mermelstein [6]) are now generally used in ASR systems as feature vectors representing a speech utterance. The feature extraction system in our phoneme recognition system implementation uses the MFCC coefficients and their corresponding Delta Coefficients, as detailed in the next chapter.

Modeling of the speech signal was later advanced by incorporating dynamic programming techniques in the pattern recognition task. Dynamic Time Warping (DTW) is a technique that was originally used to compensate for the speech variability in a spoken word in the isolated-word recognition task. DTW basically stretches or compresses the time domain of the feature vectors of the test speech signal to match the pre-existing templates of each word, in the case of the isolated word recognition task. Later, statistical models gained popularity for the task of isolated-word and continuous speech recognition. Because of the fact that the speech signal is a non-stationary stochastic process, the statistical models for a speech signal are becoming more effective for the recognition task. Perhaps the most widely used statistical model for speech word units is the Hidden Markov Model (HMM). We incorporate the HMM framework for the phoneme recognition task in this thesis.

This thesis presents an implementation of a phoneme recognizer, which is modeled by HMM. The phoneme recognizer plays a crucial role in the recognition task of a large vocabulary automatic continuous speech recognition system. Traditional algorithms for the segmentation and the recognition of the sub-word units are analyzed and a new set of algorithms is implemented for the segmentation of the phoneme boundaries and the recognition of the phoneme string. The algorithms proposed in this thesis integrate the phoneme duration statistics and the phoneme transition statistics in the recognition system.

Chapter 2

Phoneme Recognition System

The first chapter provided a brief overview of background information about ASR systems and their classification into the corresponding types based on their system properties. This chapter discusses the overall topology and the functionality of the automatic phoneme recognition system. In particular, this chapter shows the structure and the construction of the feature vector generator, for training and recognition systems. We also provide an algorithm for endpoint detection, which essentially determines the start and end points of a speech utterance. In addition several practical considerations of the ASR system construction are discussed.

2.1 Phoneme Recognition System Structure

Any pattern recognition and/or statistical inference system is comprised of two distinct phases: the training phase and the recognition phase. The phoneme recognition system implemented in this thesis consists of two layers of recognition. The first recognition layer is responsible for analyzing a set of acoustic feature vectors and determining the likelihood of the acoustic feature vector set being matched to a particular template. The second recognition layer is responsible for determining the most likely sequence of the templates, given the set of acoustic feature vectors that was extracted from the test speech utterance.

The implementation of these two recognition layers forms the two interconnected subsystems of the phoneme recognition system. The subsystem implementing the second layer uses the subsystem of the first layer for determining the best sequence of the set of templates for the given speech utterance. Each of these two subsystems has its own set of algorithms for the corresponding training and recognition phases.

There are two smaller subsystems that are responsible for the feature vector extraction and the endpoint detection of the speech utterance. These two subsystems are effectively used by the first and second layer of the recognition and training processes. The feature extraction phase is basically responsible for the generation of a set of compact feature vectors that represent the distinguishing characteristics of the speech signal. The endpoint detection system is responsible for the detection of the start and end samples of the speech utterance. Figure 2.1 illustrates the dependency graph of the feature extraction, the endpoint detection subsystems, and the training and recognition of the first and second layers.

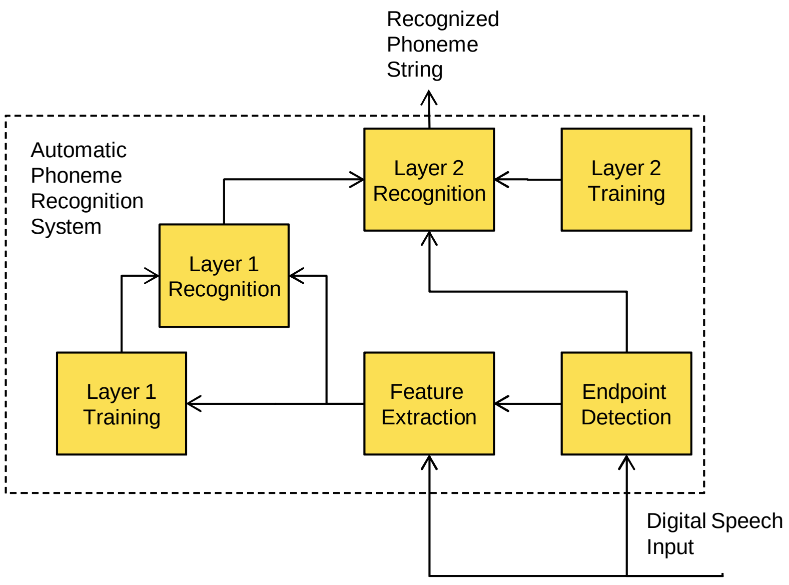

Figure 2.1: Overview of the Automatic Phoneme Recognition System.

The goal of the Automatic Phoneme Recognition (APR) system is to take the speech signal as its input and produce the text string of the phoneme sequence that corresponds to the speech utterance, as shown in Figure 2.1. The training of the phoneme recognition system is a three step process. The training of the first and second layers is done separately. First, the Feature Extraction system in Figure 2.1 is responsible for the extraction of the features from a collection of speech utterances. Second, the first layer training process generates a set of templates, each of which has a set of defining parameters. Finally, the second layer training process extracts the transition information from the phonemes and their length statistics from the speech database. After the training data has been generated, the feature extraction system generates the feature vector set for a test utterance. Then the first and second layer recognition systems together classify each of the feature vectors to one of the corresponding templates (generated in the first layer training process in Figure 2.1). Finally the system maps each of the templates to its corresponding phonemes and the second layer recognition system outputs the sequence of phonemes that was recognized from the speech signal.

The corpus of speech utterances for the training and testing (or recognition) was taken from the TIMIT speech database. The TIMIT speech database is a acoustic-phonetic speech corpus specifically designed for training and testing speaker and speech recognition systems. This database was assembled by the National Institute of Standards and Technology (NIST) sponsored by the Defense Advanced Research Projects Agency - Information Science and Technology Office (DARPA-ISTO). The TIMIT database contains 6300 sentences, from which 10 sentences are spoken by each of the 680 speakers from 8 major dialect regions of the United States. The TIMIT database is divided into two categories: a training set and a test set. The test set is comprised of speech utterances from 168 different speakers and 1344 sentences. The test set amounts to 27 % of the whole TIMIT speech database. The remaining 73 % of the TIMIT speech corpus is used for training the two layers of the phoneme recognition system.

The training and recognition processes of the first layer extract the speech information from the training and testing sets of the TIMIT database respectively and are illustrated in Figure 2.2. The training process of the first layer makes an array of feature vector sets. Each of these feature vector sets $1 \cdots N$ in Figure 2.2 pertains to a specific predefined template.

The first layer training process then generates a set of parameters (specifically the HMM parameters) for each of the predefined templates as shown in Figure 2.2. The task of the first layer recognition system is to determine the template that has the highest likelihood of being selected, given a feature vector set and the template set parameters as its input.

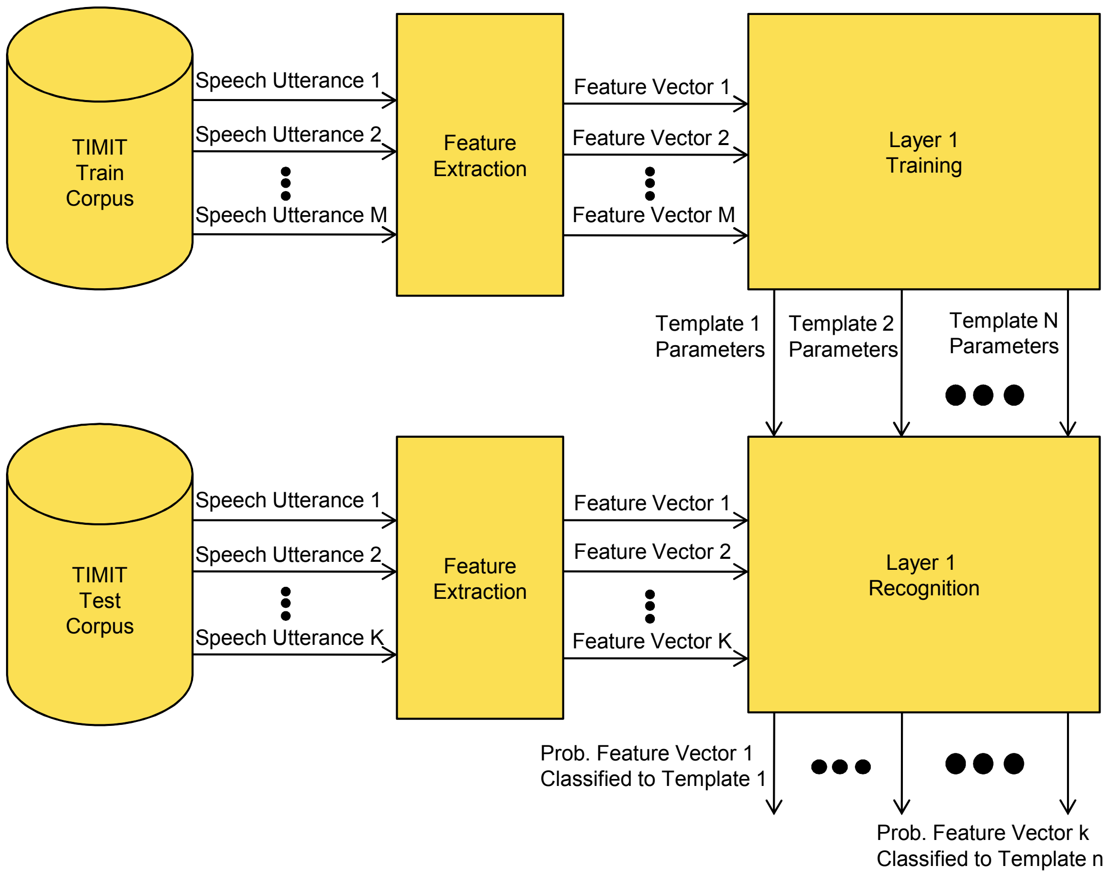

Figure 2.2: Topology of the first layer.

Figure 2.2 shows the stand-alone operation of the first layer. In the actual phoneme recognition system, the operation of the first and second layer recognition processes are interleaved. The second layer of the phoneme recognition system uses the first layer for recognizing the phoneme set as well as determining the boundaries of each phoneme in the input speech utterance. The training of the second layer provides statistical information of the length of the phonemes for phoneme segmentation, and the transition probabilities of phonemes for phoneme identification. Figure 2.3 illustrates the training and recognition aspects of the second layer.

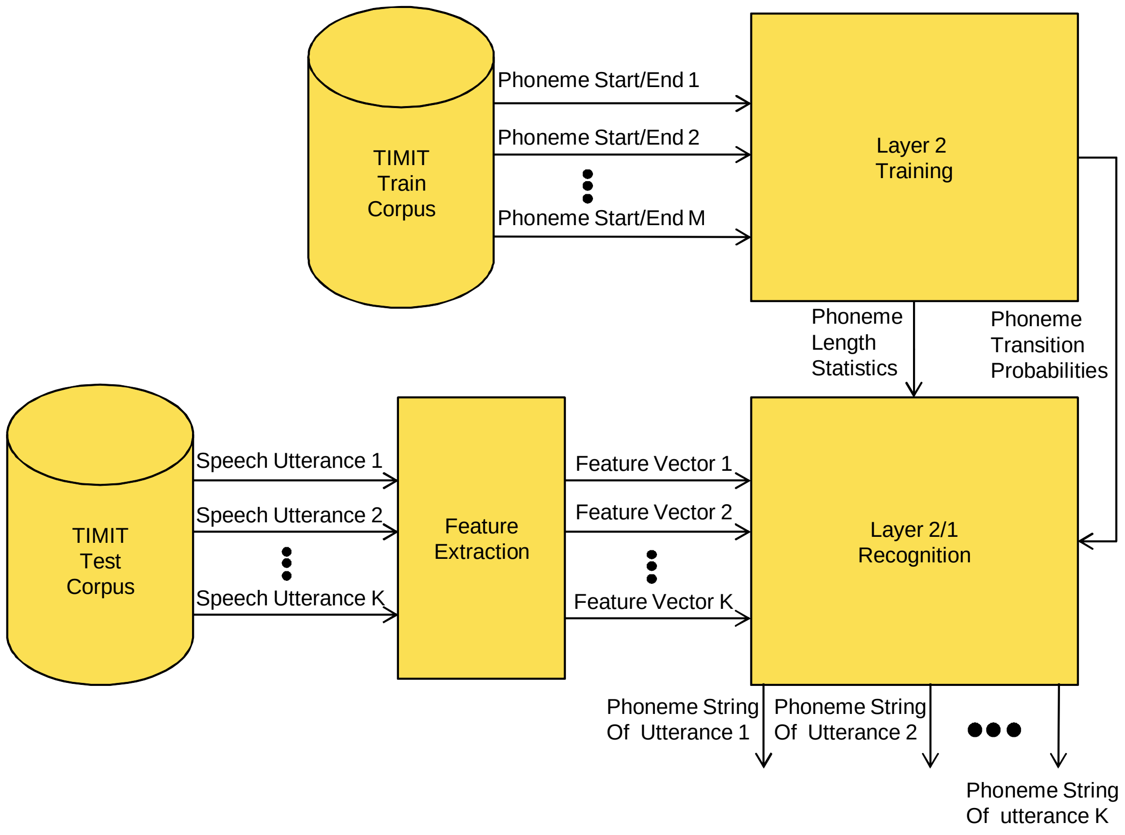

Figure 2.3: Topology of the second layer.

As seen from Figure 2.3, the combination of the first and second layer recognition systems take (as input) the array of K feature vector sets (extracted from K speech utterances from the TIMIT test corpus) as well as the phoneme length statistics and the phoneme transition probabilities. The recognition system then produces a string of segmented phoneme sets for each of the speech utterances extracted from the TIMIT test set. The training process of the second layer essentially takes the input start and endpoints of each phoneme of each speech utterance in the TIMIT training database and extracts the phoneme length statistics and the phoneme transition probabilities. The phoneme length statistics consist of the means and variances of the lengths corresponding to phonemes. Furthermore, the phoneme transition probabilities are stored in a square matrix with each element corresponding to row a and column b representing the transition probability for a transition from the b^{th} phoneme to the a^{th} phoneme.

The training and the recognition in the two layers of the automatic phoneme recognition system used in this thesis is based on two HMM models. Those are the acoustic model and the phoneme model. The acoustic model of speech models each individual phoneme based on the feature vectors gathered from the TIMIT speech training database. The phoneme model, on the other hand, enables the recognition system to look at the phoneme sequence as a whole. Thus, the first and second layers of the phoneme recognition system are based on the acoustic model and the phoneme model respectively. The HMM parameters in the acoustic model represent the distinguishing characteristics of each phoneme, while the phoneme model parameters represent the phoneme transitions and weights. We use the HMM to model the speech because the production process of the English phonemes in speech is assumed to be a discrete time-homogeneous Markov process [7]. Table 2.1 lists all the phonetic symbols in the English language.

Table 2.1: Phonetic symbols of the English language.

Phonetic Symbol	**Example Word(s)**	**Phonetic Symbol**	**Example Word(s)**
Stops:		**Semivowels and Glides:**	
b	Bee	l	Lay
d	Day	r	Ray
g	Gay	w	Way
p	Pea	y	Yacht
t	Tea	hh	Hay
k	Key	hv	Ahead
dx	Muddy, Dirty	el	Bottle
q	Bat	**Vowels:**	
Affricates:		iy	Beet
jh	Joke	ih	Bit
ch	Choke	eh	Bet
Fricatives:		ey	Bait
s	Sea	ae	Bat
sh	She	aa	Bott
z	Zone	aw	Bout
zh	Azure	ay	Bite
f	Fin	ah	But
th	Thin	ao	Bought
v	Van	oy	Boy
dh	Then	ow	Boat
Nasals:		uh	Book
m	Mom	uw	Boot
n	Noon	ux	Toot
ng	Sing	er	Bird
em	Bottom	ax	About
en	Button	ix	Debit
eng	Washington	axr	Butter
nx	Winner	ax-h	Suspect

There is a set of phonemes that do not occur in the lexicon and are only hand-segmented in the transcriptions of the TIMIT training and testing corpus. Table 2.2 lists these phonetic symbols and their corresponding description.

Table 2.2: Phonetic symbols used in transcription only.

Symbol	**Description**
bcl	closure symbol for the stop b
dcl	closure symbol for the stop d and jh
gcl	closure symbol for the stop g
pcl	closure symbol for the stop p
tck	closure symbol for the stop t
kcl	closure symbol for the stop k
tcl	closure portion of ch
pau	pause
epi	epiphanic silence

In Table 2.2, the closure symbols for the stops are the closure intervals of the stops that are distinguished from the stop release and the epiphanic silence is typically found between a fricative and a semivowel or between a fricative and a nasal. Epiphanic silence can also be found at the non-speech events at the beginning and the end of the speech signal. The symbols listed in Tables 2.1 and 2.2 comprise the set of phonemes that are considered for the phoneme recognition task. The actual set of phonemes are chosen according to their weights after they have been extracted from the TIMIT training database.

The training data set used to train the two HMM models in the automatic phoneme recognition system contains a collection of 4956 distinct speech utterances coming from 462 different speakers. Due to the number of speech utterances and the number of speakers there is a significant variation in the pronunciation, length, and pitch of the phonemes. To compensate for these variations, the lists of phonemes in Tables 2.1 and 2.2 are further divided into subgroups.

There are two kinds of variations that our phoneme recognition system tries to compensate for: speaker variability and context variability. Speaker variability refers to the variations in the phoneme characteristics that is generated by a training data set containing a large number of speakers. Context variability, on the other hand, comes from a speaker data set containing a large lexicon. Research has shown that there is a significant difference in the pitch of the speech signal and the pronunciation of the phonemes when the speech utterance comes from a male versus from a female. Thus it makes sense to train the different phoneme models for males and for females [8] [9]. Also, research has shown that acoustic characteristics of the phoneme depend on the following and the preceding phonemes [10]. In particular, the largest differences in phoneme characteristics have been observed between phonemes preceded by a vowel and phonemes preceded by a non-vowel phoneme [11]. Thus, the training data set is further clustered into phonemes that are preceded by a vowel and phonemes that are preceded by a non-vowel in the speech utterances of the TIMIT training corpus. The set of vowels is part of the list in Table 2.1.

2.2 Feature Vector Extraction

The Mel-Frequency Cepstral Coefficients (MFCC) of a speech signal are commonly used for obtaining good recognition results in speech and speaker recognition tasks. The MFCC are used extensively in the speech/speaker recognition literature for two reasons. Firstly, MFCC have a low number of dimensions, which effectively avoids the curse of dimensionality for the recognizers. Secondly, MFCC closely relate to the biology of the filtering performed in the human ear. For these reasons we have included the MFCC extraction in our feature vector extraction system. There is a total of eight steps involved in obtaining the MFCC feature vector for a frame of speech. These steps are outlined below and shown in Figure 2.4.

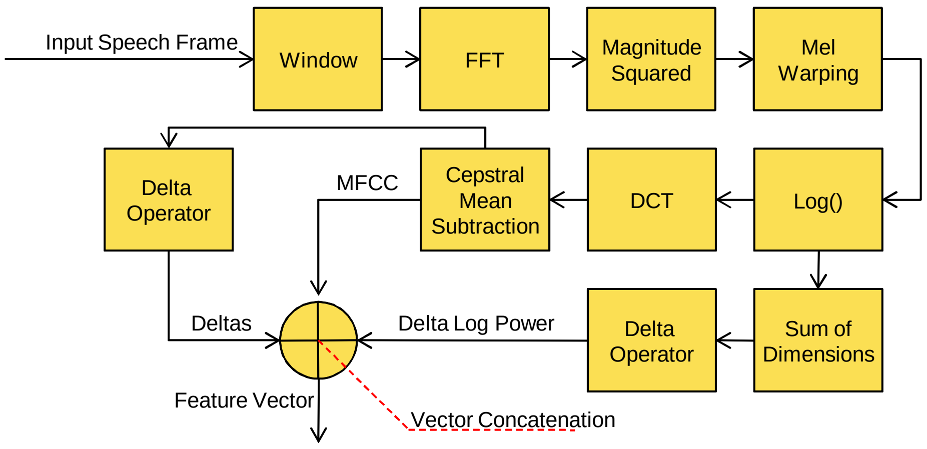

Figure 2.4: Feature Extraction System block diagram.

The steps in obtaining MFCC are the following.

1. **Windowing the signal.**

   A frame of a signal is defined as a sequential (in time) aggregation of a sampled signal. Thus the analog signal is first sampled (generally by means of an Analog to Digital Converter) then windowed as expressed in the following equation:

$$x_n(t) = w(t)s\left(nS_F + t\right) \quad for \quad t = 0, 1, \ldots, L-1 \quad n = 0, 1, \ldots, N-1 \tag{2.1}$$

   where $s(t)$ is the sampled signal, n is the index for the particular frame, S_F is the frame step size (the number of samples between the start indexes of two consecutive frames), L is the length of the window and N is the number of consecutive frames in the speech utterance. For this thesis work the window function used is the Hamming Window, for which $w(t)$ is defined by:

$$w(t) = \alpha - \beta \cos\left(\frac{2\pi(t-1)}{L-1}\right) \tag{2.2}$$

   for two constants $\alpha = 0.54$ and $\beta = 0.46$ [12]. Typically there is an overlap between the length of the window and the start of the next frame. Thus the window length L is generally greater than the frame step size S_F.

2. **FFT of the windowed frame.**

The Fast Fourier Transform of the frame in (2.1) is functionally equivalent to the Discrete Fourier Transform of the windowed signal $x_n(t)$ for the n^{th} frame in (2.1), and is given by:

$$X_n(m) = \sum_{t=0}^{L-1} x_n(t) \exp\left(\frac{-i2\pi mt}{F}\right) \quad m = 0, 1, \ldots, F-1 \tag{2.3}$$

where F is the length of the FFT vector (typically = 256) and m is the indexing variable for the FFT. The FFT for the n^{th} frame of the speech signal is denoted by $X_n(m)$.

3. **Magnitude Squared of the frame FFT.**

The Magnitude Squared of the complex signal $X_n(m)$ in (2.3) associated with the n^{th} frame is given by:

$$|X_n(m)|^2 = X_n(m)X_n'(m) \quad m = 0, 1, \ldots, F-1 \tag{2.4}$$

The phase of the complex signal $\angle X_n(m)$ is discarded in this step, because it generally is considered not to contain any distinguishing information regarding the speech signal. The human ear also discards the phase of the signal for the recognition and identification tasks and uses the phase of a speech signal mainly for distinguishing the location of the source of a speech signal.

4. **Mel Warping of the FFT Magnitude.**

This step of the MFCC transform process is primarily used to further reduce the dimensionality of the feature vector. The Mel Warping step of the magnitude squared signal $|X_n(m)|^2$ in (2.4) applies a magnitude filter bank and averages the power in each of the frequency bands of the signal $|X_n(m)|^2$. Let $f_j(m)$ represent the weights of filter j in the filter bank. The filter magnitudes of each filter $f_j(m)$ in the filter bank correspond to a triangular filter [13]. Furthermore, the bandwidth of $f_j(m)$, defined by the low cutoff and the high cutoff frequency of the j^{th} triangular filter overlaps with its neighboring $(j-1)^{th}$ and/or $(j+1)^{th}$ filter(s). The low and high cutoff frequencies are distributed equally on the Mel-Scale. Let the function $f_{mel} = MEL(f)$ represent the mapping between the regular frequency and the Mel frequency and this mapping is given by:

$$f_{mel} = MEL(f) = 2595 \log\left(1 + \frac{f}{700}\right) \tag{2.5}$$

Figure 2.5 shows an example of the overlapping triangular filter bank weights distributed over the Mel scale on the frequency axis. Let $m = MEL_{bin}(f)$ denote the frequency bin m associated with the Mel frequency f_{mel}. Also, let J denote the total number of filters used in the filter bank and F_s represent the sampling frequency used to acquire the speech utterance. Then the bandwidth of the j^{th} filter in the filter bank, in terms of the number of FFT bins, is given by:

$$Bw_j = MEL_{bin}^{-1}\left(\frac{F_s}{J+2}(j+2)\right) - MEL_{bin}^{-1}\left(\frac{F_s}{J+2}j\right) \tag{2.6}$$

The average power for each filter in the filter bank is then given by:

$$X_n^j = \frac{1}{Bw_j}\sum_{m=0}^{Bw_j}\left[f_j(m)\left|X_n\left(MEL_{bin}^{-1}\left(\frac{F_s}{J+2}j\right)+m\right)\right|^2\right] \tag{2.7}$$

Typically $|X_n(m)|^2$ is a vector of dimension 256 and this step generally reduces this number to $J = 35$ dimensions by applying a filter bank containing 35 band-pass filters.

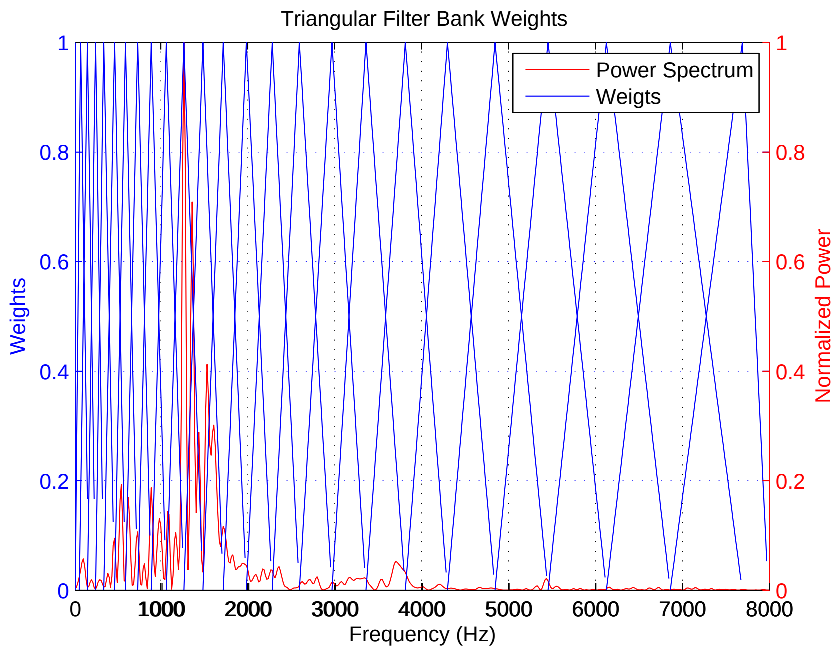

Figure 2.5: Overlapping triangular filter bank weights with cutoff frequencies distributed on the Mel frequency scale.

5. **Logarithm of the Mel-Warped FFT magnitude squared signal.**

   The logarithm of the signal X_n^j in (2.7) is given by:

$$L_n^j = \ln X_n^j \tag{2.8}$$

6. **Discrete Cosine Transform of the Log of Mel-Warped FFT magnitude squared signal.**

   This step of the MFCC transform further reduces the dimensionality of the feature vector by generally eliminating dimensions 13 and higher, leaving a 12 dimensional acoustic feature vector for each frame in the speech signal. The Discrete Cosine Transform of the signal vector L_n^j in (2.8) is given by:

$$x_n^c = C \cdot L_n^j \tag{2.9}$$

where the elements in the matrix C are defined as:

$$C_{p,l} = \sqrt{\frac{2}{D_{mfcc}}} \cos\left(\frac{\pi p}{D_{mfcc}}(l - \upsilon)\right) \tag{2.10}$$

where the constant D_{mfcc} is the number of rows in the matrix C in (2.9). The variables p and l are the indexing variables of the matrix C in (2.9) and $\upsilon = 0.5$ is a constant. Thus the matrix C is a $D_{mfcc} \times J$ matrix where the number of columns J corresponds to the length of the vector L_n in (2.8) for each frame n (and is equal to the number of filters used in the filter bank).

7. **Cepstral Mean Subtraction.**

   In this step the mean of the feature vectors x_n^c is subtracted from each feature vector in an effort of making the feature vectors more robust. This process is called Cepstral Mean Subtraction (CMS) since we are subtracting the mean of the Cepstral coefficients from the Cepstral coefficients. Thus the final MFCC coefficients are given by:

$$MFCC_n = x_n^c - \frac{\sum_{n=1}^{N} x_n^c}{N}. \tag{2.11}$$

8. **Calculating Deltas of the signal.**

   The Delta Coefficients of each frame in the speech signal also have dimension 12. These coefficients are given by:

$$\triangle MFCC_n = \frac{\sum_{r=1}^{R} r \cdot (MFCC_{n+r} - MFCC_{n-r})}{2\sum_{r=1}^{R} r^2} \tag{2.12}$$

   where R is a parameter, which has a value of 3 [14]. In the same manner the acceleration parameters (not included in the feature vectors) are given by:

$$\triangle a_n = \frac{\sum_{r=1}^{R} r \cdot (\triangle MFCC_{n+r} - \triangle MFCC_{n-r})}{2\sum_{r=1}^{R} r^2} \tag{2.13}$$

   This step also computes the Delta Log Power coefficients, which is the result of the Delta operation being applied to the sum of the log of the Mel-warped FFT magnitude squared signal (summed over all of its filter banks). Hence, one dimensional Log power coefficients are given by:

$$LP_n = \sum_{j=1}^{J} L_n^j \tag{2.14}$$

Delta Log Power coefficients are one dimensional coefficients and represent the change in the log power of the speech signal. They are given by:

$$\triangle LP_n = \frac{\sum_{r=1}^{R} r \cdot (LP_{n+r} - LP_{n-r})}{2\sum_{r=1}^{R} r^2} \tag{2.15}$$

The final feature vector for each frame in the speech signal used in the phoneme recognition system (the output of the Feature Extraction system in Figure 2.1) is a 25 dimensional feature vector comprised of the concatenation of the 12-D MFCC coefficients obtained in step 7, the 12-D delta coefficients (2.12) and the 1-D delta log-power coefficient (2.15) (both obtained in step 8). It is important to note the probability distributions of the speech signal before and after the MFCC transformation (step 7). If the speech signal is modeled to have a Gaussian distribution, then the Cepstral features are Rayleigh distributed, the Mel-Warped Power Cepstral features (step 4) will be Chi-Square distributed and the log Cepstrum and the MFCC features can be accurately modeled as a mixture of multivariate Gaussian distributions [13]. This gives the advantage for modeling the MFCC domain signal as a Gaussian Mixture Model (GMM).

2.3 Endpoint Detection

The Endpoint Detection system in Figure 2.1 is responsible for isolating the speech data from the non-speech data in the speech utterance. The input and output of the Endpoint Detection system is the speech signal and the truncated speech signal respectively. The Endpoint Detection system effectively eliminates the non-speech signal from the beginning and the end of the speech utterance in the TIMIT speech database. Thus the output of the Endpoint Detection system contains only the speech signal. The endpoint detection algorithm implemented for the phoneme recognition system is a slight modification of the algorithm proposed by Rabiner and Sambur [15].

The implemented endpoint detection algorithm extracts the endpoint of the speech utterance based on two measures: the energy of the speech signal and the zero-crossing rate of the speech signal. The endpoint of the signal is in first instance based on the energy of the signal, which is the primary indicator of the start and finish points of the speech content, since it is assumed that the noise energy in the speech signal is considerably lower than the speech energy. However, the energy threshold gives a constrained measure of the endpoint of the speech utterance because it does not take into account the distinction between voiced and unvoiced speech. After the energy thresholds are set we compute the zero-crossing rate over a certain interval where the energy of the speech signal is within the set energy thresholds. The zero-crossing rate has been shown to provide a relatively good measure for distinguishing between voiced and unvoiced speech [15].

The endpoint detection algorithm is implemented by first computing the energy approximation of the signal for each speech sample in a 10 ms interval around each sample point. Thus the energy approximation of the speech signal is given by,

$$E(t) = \begin{cases} \sum_{i=1}^{t+S_F/2} |s(i)| & for \quad 0 < t \leq S_F/2 \\ \sum_{i=t-S_F/2}^{t+S_F/2} |s(i)| & for \quad S_F/2 < t \leq T - S_F/2 \\ \sum_{i=t-S_F/2}^{T} |s(i)| & for \quad T - S_F/2 < t \leq T \end{cases} \tag{2.16}$$

where T is the number of sample points in the speech utterance, $s(i)$ is the sampled speech signal and S_F is the frame step size as mentioned in Section 2.2. The energy thresholds are computed based on the maximum and the minimum energy. There are two energy thresholds that are being considered for the estimation of the speech utterance endpoints. A lower energy threshold and an upper energy threshold. These are computed as follows:

$$ITL = \min\left[0.03 \cdot \{max\left(E(n)\right) - \min\left(E(n)\right)\} + \min\left(E(n)\right), 4 \cdot \min\left(E(n)\right)\right] \tag{2.17}$$

$$ITU = 5 \cdot ITL \tag{2.18}$$

where ITL is the lower energy threshold and ITU is the upper energy threshold. The start and end points of the speech utterance are estimated based on these two thresholds. The start of the speech utterance is first approximated by finding the sample point $s(t)$ having the largest index t for which the energy $E(t)$ is less than the energy of the first sample violating the upper threshold ITU and greater than the energy of the first sample violating the lower threshold ITL. Likewise, the end point of the speech utterance is first approximated by finding the sample $s(t)$ having the smallest index t for which the energy $E(t)$ is greater than the energy of the last sample violating the lower threshold ITL and smaller than the energy of the last sample violating the upper energy threshold ITU. In summary the start and end points are given by:

$$N_1 = \max_t \left(\operatorname*{argmax}_t \left(E(t) < ITL \right) \right) \cap \left(1, \operatorname*{argmin}_t \left(E(t) > ITU \right) \right) \tag{2.19}$$

$$N_2 = \min_t \left(\operatorname*{argmax}_t \left(E(t) > ITU \right), P \right) \cap \left(\operatorname*{argmin}_t \left(E(t) < ITL \right) \right) \tag{2.20}$$

where, N_1 is the initial approximation of the start point of the speech signal and N_2 is the initial approximation of the end point of the speech signal. The lower energy threshold is always less than approximately three percent of the speech energy range. The upper threshold is always less than approximately 15% of the energy range, since the lower energy threshold is less than approximately 3% of the speech energy range and the upper energy threshold is five times the lower energy threshold (as seen from (2.17) and (2.18)). This is illustrated on a speech energy signal in Figure 2.6.

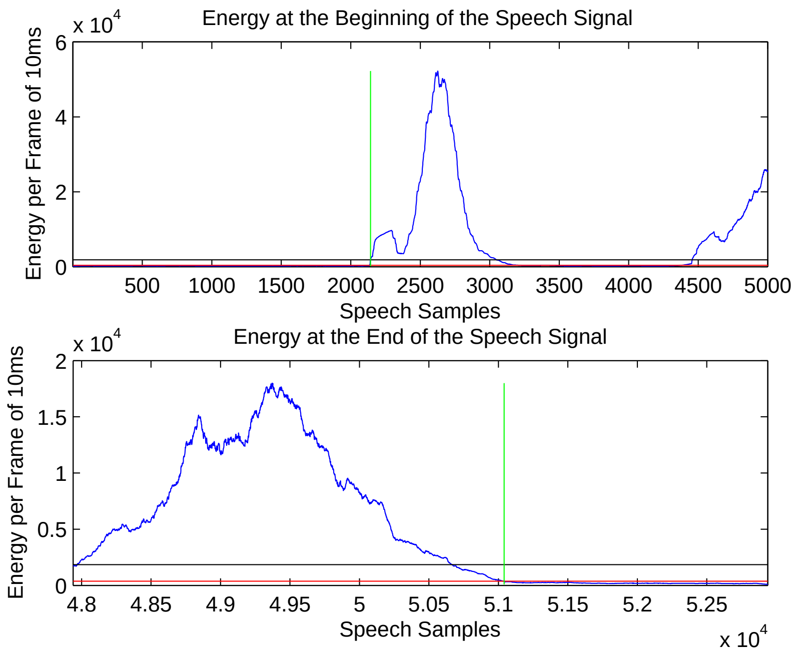

Figure 2.6: Beginning and ending of the speech signal energy and the corresponding energy thresholds.

The black and red lines in Figure 2.6 correspond to the upper and lower energy thresholds respectively, while the green line in the top figure in Figure 2.6 represents the approximation to the start point of the speech signal and the green line in the bottom figure in Figure 2.6 represents the approximation to the end point of the speech signal. The endpoint approximation of the speech signal based only on the energy of the signal is too constrained since it doesn't take into account the fricatives (especially the weak fricatives that have low energies) at the beginning and end of a speech utterance. However it is safe to assume that the speech signal is contained within the interval (N_1, N_2). Thus the endpoint detection algorithm proceeds with examining the zero-crossing rates over a 50 ms interval before N_1 and a 50 ms interval after N_2. The final endpoints of the speech signal are determined based on the number of times the zero-crossing threshold rate was violated within the interval of examination before and after N_1 and N_2 respectively. The zero-crossing rates are determined around a given point by counting the number of times the speech signal changes sign in an interval of 10 ms about the given point. Thus the zero-crossing rate is given by:

$$ZCR(t) = \begin{cases} \sum_{i=2}^{t+S_F/2} \delta\left(sign\left(s(i-1)\right) \neq sign\left(s(i)\right)\right) & for \quad 0 < t \leq S_F/2 \\ \sum_{i=t-S_F/2}^{t+S_F/2} \delta\left(sign\left(s(i-1)\right) \neq sign\left(s(i)\right)\right) & for \quad S_F/2 < t \leq T - S_F/2 \\ \sum_{i=t-S_F/2}^{T} \delta\left(sign\left(s(i-1)\right) \neq sign\left(s(i)\right)\right) & for \quad T - S_F/2 < t \leq T \end{cases} \tag{2.21}$$

where $\delta(A \neq B)$ returns one if and only if A is not equal to B. Otherwise $\delta(A \neq B)$ returns zero. The threshold for the zero-crossing rate is computed by first gathering statistics on the zero-crossing rate for silence. It is assumed that the first and the last 60 ms of speech will be silence (or a small non-speech noise). The mean and standard deviation is computed for the zero-crossing rates from sample $S/2$ to sample $5 * S + S/2$ and from sample $T - 5 * S - S/2$ to sample $T - S/2$. The zero-crossing rate threshold is given by:

$$IZCT = \min(105, \overline{ZCR} + \sigma_{ZCR}) \tag{2.22}$$

where $\overline{ZCR}$ and σ_{ZCR} are the mean and standard deviation of the zero-crossing rates respectively for the small noise before and after the speech signal. Figure 2.7 shows the zero-crossing rates for the speech signal and the corresponding threshold. From Figure 2.7 we see that the zero-crossing rate is relatively large for noise at the beginning and end of the speech utterance and small for voiced speech. The red line in Figure 2.7 corresponds to the zero-crossing rate threshold.

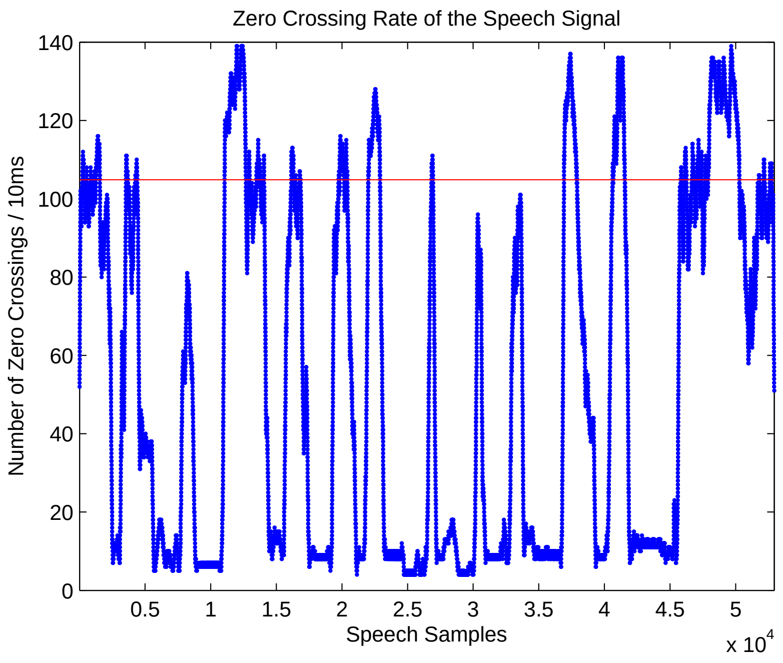

Figure 2.7: Zero crossing rate plot and the corresponding zero-crossing rate threshold.

The final start and end points of the speech signal are determined based on the number of times the zero-crossing rate $ZCR(t)$ violates the zero-crossing threshold $IZCT$ on the 50 ms interval before N_1 and 50 ms interval after N_2. If the number of the zero-crossing rate threshold violations in the 50 ms interval before N_1 is greater than 30 then the first zero-crossing rate violation is picked as the start of the speech signal, otherwise the start point of the speech remains N_1. Likewise if the number of zero-crossing violations in the 50 ms interval after N_2 is greater than 30 then the last zero-crossing rate violation is picked as the end of the speech signal, otherwise the end sample of the speech remains N_2. Figure 2.8 shows the result of the endpoint detection algorithm.

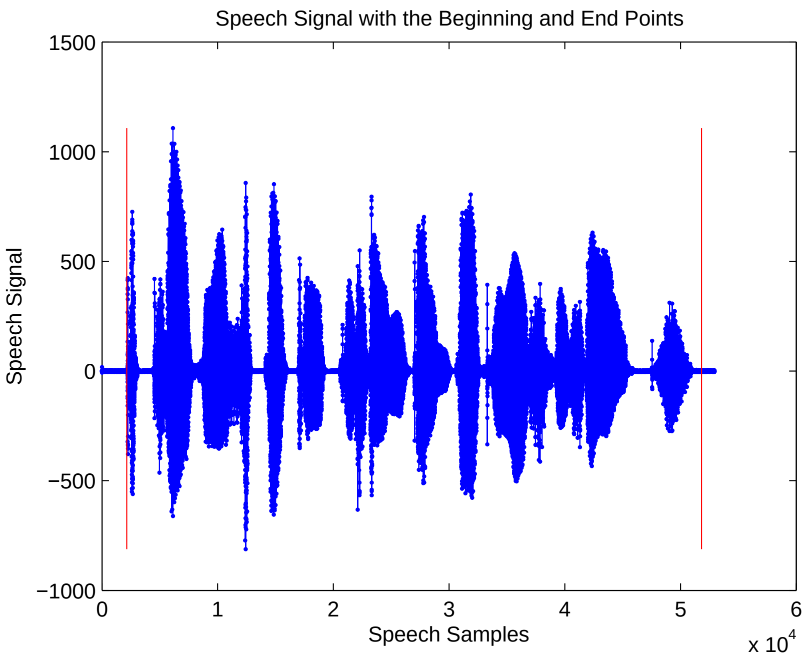

Figure 2.8: Speech with its detected start and end points.

Chapter 3

Hidden Markov Models

The automatic phoneme recognition system implemented in this thesis is based on two continuous Hidden Markov Models. The left-to-right HMM is used for detecting any distinguishing characteristics in the acoustical structure of the phoneme. The ergodic Segmental HMM is used for integrating the phoneme length statistics and the phoneme transition statistics in the phoneme recognition task. This chapter discusses the training and recognition of the HMM models and their application to phoneme recognition.

3.1 HMM Model Elements

The HMM models two stochastic processes: the observation sequence and the hidden state sequence, and the relationship between these two processes. The observation sequence is a set of event outcomes occurring in a sequential manner. For example, the sequence of outcomes from an unfair coin tossing experiment or (for the case of speech) the sequence of feature vectors generated by the feature extraction system for each time frame of speech. However, the hidden state sequence can only be inferred by analyzing the statistics of the observation sequence. For example, in an unfair coin tossing experiment, the observation sequence would be the outcome of the coin tossing (heads of tails) and the hidden state sequence would be the unfair coin being tossed (one of a number of unfair coins). For the case of speech, the hidden sequence is the sequence of phonemes spoken, which is inferred from the sequence of feature vectors.

We now examine composition and structures of the HMM and its operation. The HMM model represents a set of parameters describing the relationship between the observation sequence and the hidden state sequence. At any given time an observation based on an HMM can be labeled as being associated with one of the S discrete hidden states $[1, 2, \ldots, S]$. The time index at any given time is represented by n and the random variable representing the hidden state at time n is denoted by q_n. The sequence of hidden states in a system modeled with an HMM is assumed to be a stochastic Markov process. A stochastic process is a first order Markov process if and only if the probability of a future state at any given time depends only on its present state, i.e. on none of the past states. Thus,

$$P(q_{n+1}|q_n, q_{n-1}, \ldots, q_1) = P(q_{n+1}|q_n) \tag{3.1}$$

Since, in an HMM model, each of the states at a given time n can only be associated with one of the S possible hidden states, there is a finite number of probability values possible for the right hand side of (3.1). These probabilities are represented in a state transition matrix, which is given by:

$$\mathbf{A} = \begin{bmatrix} P(q_n = 1|q_{n-1} = 1) & P(q_n = 1|q_{n-1} = 2) \cdots & P(q_n = 1|q_{n-1} = S) \\ P(q_n = 2|q_{n-1} = 1) & P(q_n = 2|q_{n-1} = 2) \cdots & P(q_n = 2|q_{n-1} = S) \\ \vdots & \vdots & \vdots \\ P(q_n = S|q_{n-1} = 1) & P(q_n = S|q_{n-1} = 2) \cdots & P(q_n = S|q_{n-1} = S) \end{bmatrix} \tag{3.2}$$

Thus each of the elements in the state transition matrix A is denoted by a_{ij} and is given by:

$$a_{ij} = P(q_n = i|q_{n-1} = j) \tag{3.3}$$

Since the elements in the state transition probability matrix represent probabilities of events, they are non-negative numbers. Due to the fact that there is only one state transition at a given time index the columns of A must sum to unity. Thus:

$$0 \leq a_{ij} \leq 1 \quad \forall\, i \;\in\; \{1:S\}, \quad \forall\, j \in\; \{1:S\} \tag{3.4}$$

and

$$\sum_{i=1}^{S} a_{ij} = 1 \tag{3.5}$$

Let the observation at time n be denoted as O_n. The HMM also models the probabilities of the hidden states $\{q_n\}_{n=1}^{N}$ given the set of observations $\{O_n\}_{n=1}^{N}$. Let v_n denote the random variable representing the observation at time n. For a simple HMM there is a single probability measure for the probability of the state given the observation at each time index n. The probability of a observation vector given the hidden state q_n at time index n is called the emission probability and is given by:

$$b_n^i = P(v_n = O_n | q_n = i) \tag{3.6}$$

Also let B denote the set of emission probabilities given by:

$$\mathbf{B} = \{b_n^i\} \; for \; i = 1, 2, \ldots, S \tag{3.7}$$

Finally, the HMM models the probability of the first hidden state $P(q_1 = i)$ by the initial state probabilities given by:

$$P(q_1 = i) \equiv \pi_i \tag{3.8}$$

In summary, a system based on an HMM can be fully described by the initial state probability vector π_i, the state transition matrix A and the set of emission probability distributions B. Thus an HMM model λ is represented by:

$$\lambda = (A, B, \pi) \tag{3.9}$$

Let π^n denote the vector of S probabilities of the hidden states at time index n. Thus:

$$\pi^n = \begin{bmatrix} P(q_n = 1) \\ P(q_n = 2) \\ \vdots \\ P(q_n = S) \end{bmatrix} \tag{3.10}$$

The state probability vector can also be written as:

$$\pi^n = A \cdot \pi^{n-1} = A^{n-1} \cdot \pi^1 \tag{3.11}$$

where π^1 in (3.11) represents the initial probabilities of the hidden states.

3.2 HMM Model Types

There is a number of different types of HMMs, each of which has its own unique characteristics. These HMM models differ from each other based on two properties: the structure of the HMM and the type of the emission probability density function. The emission probability density function defines whether or not the emission probabilities are given by a probability density function (of the observations) or a probability mass function of the quantized observation regions. As a result, the emission probability density function defines if the observations follow a continuous or a discrete distribution. The structure of the HMM can be defined by imposing constraints on the hidden state transition matrix A. There are two types of HMM structures that are used in the phoneme recognition system, the left-to-right HMM and the ergodic HMM. There is one other variation to the structure of one of the HMMs used in this thesis which is called a Segmental HMM. The Segmental HMM generalizes the left-to-right HMM by allowing multiple observations to be associated with a single hidden state. All these types of HMM are described below.

3.2.1 Continuous vs. Discrete

HMMs can be divided into two distinct categories: continuous HMM and discrete HMM. For the discrete HMM the observation at each time index n can only take on a finite number of values. Thus the observation space is divided into a finite number of subspaces the union of which is the observation space. With this, there are only a finite number of emission probabilities for each hidden state. The quantization of the observation space is typically carried out by means of Vector Quantization (VQ). The discrete HMM generally has the advantage of low computational complexity, but it may suffer from inaccurate results. Research has shown that phoneme recognition systems modeled with continuous HMMs yield higher recognition rates than phoneme recognition systems modeled with discrete HMMs [16]. Thus we use the continuous HMM to model the set of English phonemes.

In a continuous HMM design the observations are assumed to be continuous. In other words there is no constraint on the values that the observations can take. The emission probabilities, in this case, are defined by probability density functions (pdf). We have used a Gaussian Mixture Model (GMM) to represent the emission probabilities for the HMM model representing the first layer of the phoneme recognition system. In this model the emission probabilities are defined by:

$$b_{q_n}^{On} = P(v_n = O_n | q_n = i) = \sum_{k=1}^{K} c_{ik} N(O_n; \mu_{ik}, \Sigma_{ik}) \tag{3.12}$$

where c_{ik} are the weights of each of the Gaussian pdf's in the GMM, $N(O_n; \mu_{ik}, \Sigma_{ik})$ denotes a multivariate Gaussian pdf with mean vector μ_{ik}, covariance matrix Σ_{ik} and observation O_n at time index n for hidden state i and mixture k. The corresponding probability of the multivariate Gaussian is given by:

$$N(O_n; \mu_{ik}, \Sigma_{ik}) = \frac{1}{(2\pi)^{D/2}(\det(\Sigma_{ik}))^(1/2)} e^{-((O_n-\mu_{ik})^T \Sigma_{ik}^{-1}(O_n-\mu_{ik}))/2} \tag{3.13}$$

where D is the dimensionality of the feature vectors (which in this thesis is 25). Since the emission probabilities come from a pdf, the GMM mixture coefficients have to form a convex combination, i.e. be nonnegative and satisfy the constraint:

$$\sum_{k=1}^{K} c_{ik} = 1 \tag{3.14}$$

3.2.2 Variations in HMM Structure

There are several types of HMMs that differ from each other based on the constraints on the allowable transitions of the hidden states of the HMM. The two HMMs used in this thesis (for the two layers of recognition) differ from each other structurally. The HMM constructed for the recognition of the first layer is a left-to-right HMM. On the other hand, the HMM constructed for the second layer of recognition is an ergodic HMM. In an ergodic HMM there is no constraint on the direction of the transitions between the hidden states and thus a transition may occur between any two hidden states. Thus all of the elements in the state transition probability matrix may be nonzero. In the left-to-right HMM there is only one valid starting and ending hidden state and the transitions are only allowed from the preceding hidden state (starting from the first valid state) to the current state in a left-to-right fashion. Each element a_{ij} in the state transition probability matrix is nonzero only for $i = j$ or $i = j + 1$ for $j = 1, 2, \ldots, S$ (for diagonal and first superdiagonal).

The state transition probability matrix of an HMM can be represented graphically using a directed graph model. In this graph model each node represents a hidden state and the arrows represent allowable transitions between the hidden states (or nonzero entry in the hidden state transition probability matrix). Figure 3.1 shows the graph model for the state transitions in a 4-state ergodic (Figure 3.1a.) and in a 4-state left-to-right HMM (Figure 3.1b.).

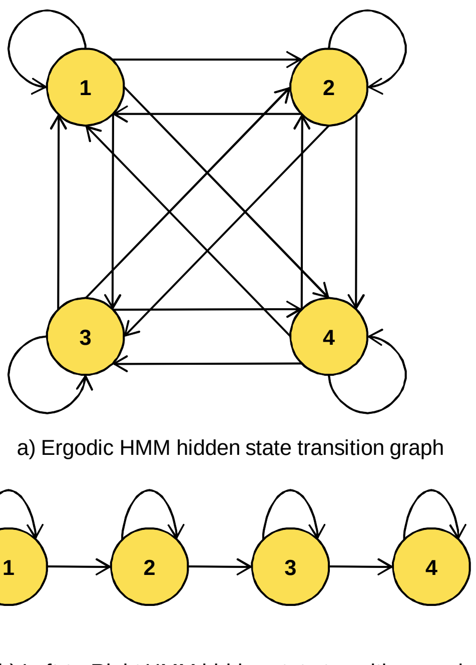
1
2
3
4
a) Ergodic HMM hidden state transition graph
1
2
3
4
b) Left-to-Right HMM hidden state transition graph

Figure 3.1: Ergodic and left-to-right HMM hidden state transition graphs.

As the top of Figure 3.1 illustrates, the ergodic HMM state transition graph is fully connected and thus any transition between two hidden states is possible. The bottom of Figure 3.1 shows the left-to-right HMM state transition structure, where each state can either transition back to itself or to the state on its immediate right.

The implementation of the second layer recognition system is based on a variant of the ergodic HMM in Figure 3.1 and is called a Segmental HMM. The variation in the structure of the HMM is a generalization of the probabilistic relationship between the observation sequence and the hidden state sequence. The Segmental HMM allows a hidden state to ”produce” multiple observations. Thus the emission probabilities of each hidden state are associated with several observations, as opposed to being associated with a single observation in the case of a regular HMM. During the recognition phase of the Segmental HMM the observation sequence is divided into segments and each segment of observations is related to a single hidden state (hence the name Segmental HMM). Figure 3.2 illustrates the structure of the Segmental HMM.

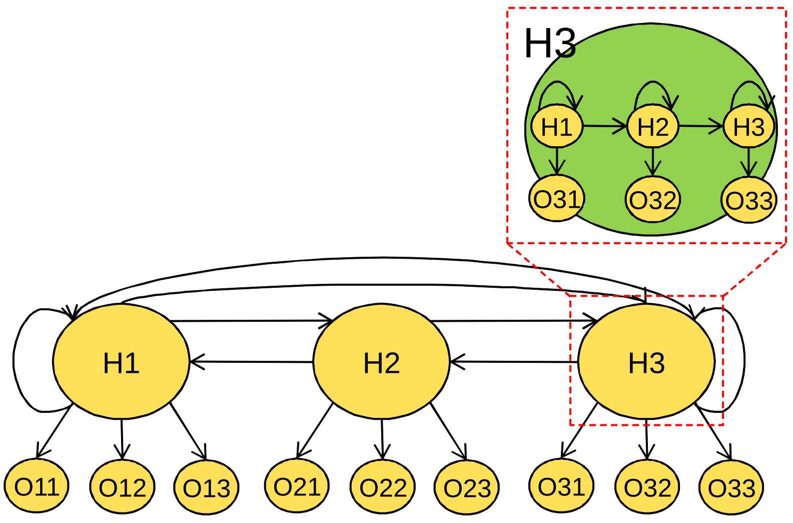

Figure 3.2: Ergodic Segmental HMM hidden state transition and observation graph.

Figure 3.2 is an example of a 3-state ergodic Segmental HMM. As we see from Figure 3.2 the graph corresponding to the hidden states of the HMM is fully connected, indicating that the HMM is an ergodic HMM. Also, every hidden state Figure 3.2 corresponds to a left-to-right HMM model. As such, every hidden state in Figure 3.2 has multiple observations produced by its emission probability. The emission probabilities in this case are defined over a multivariate probability density space where the dimensions of each emission probability space depends on the number of observations assigned to a hidden state. The observation sequence is segmented and each segment is assigned to one hidden state during the recognition phase of the Segmental HMM system. The Segmental HMMs - and their applications to speech recognition tasks - are not new in the literature. Different forms and types of Segmental HMMs have been applied to continuous syllable recognition [17], and to glottal pulse segmentation from time domain speech signals [18]. Also a Segmental HMM based on polynomial trajectories has been applied to continuous speech recognition [19]. More publications on the use of Segmental HMMs on speech recognition tasks can be found here [20] [21]. However, the form and method of the application of the Segmental HMM model topology to the task of phoneme sequence recognition is unique in this thesis.

3.2.3 Three basic problems of HMM

For any HMM type mentioned in the previous section, there are three basic problems that must be solved during the training and recognition phases of the HMM. These three tasks are:

1. Given the observation sequence $O_1, O_2, \ldots, O_N$ and the model $\lambda = (A, B, \pi)$ efficiently compute the probability of the observation sequence given the model.

2. Given the observation sequence $O_1, O_2, \ldots, O_N$ and the model $\lambda = (A, B, \pi)$ efficiently compute the sequence of hidden states of the model which maximizes the likelihood of the observation sequence and the model.

3. Given the observation sequence $O_1, O_2, \ldots, O_N$ adjust the model parameters of the model $\lambda = (A, B, \pi)$ to maximize the likelihood of the observation sequence and the model.

The solution to the first problem is used during the training phase of the system, specifically in the implementation of the training algorithm of the HMM in the first layer of recognition. Problem one is efficiently solved by the Forward-Backward algorithm. The solution to the second problem is used in the recognition of both HMMs in this thesis. The second problem is efficiently solved by the Viterbi Search algorithm. The Viterbi Search algorithm is similar to the Forward-Backward algorithm as will be seen, in Sections 3.3 and 3.4. The solution to the third problem is used in the training of the HMMs. There are several good algorithms for the solution of problem three. The algorithm used in this thesis was originally proposed by Baum and his colleagues and is called the Baum-Welch algorithm. These algorithms are described in detail in the next sections. We start with the problem of recognition and the solution to the second fundamental problem of HMM.

3.3 Viterbi Search Algorithm

The Viterbi Search algorithm presents an easy and efficient way of finding a single best hidden state sequence for the given model and the observation sequence. The algorithm uses Dynamic Programming to find the best ”path” in the search space of possible hidden state sequences that produces the maximum likelihood of the observation set, given the hidden state sequence and the HMM model. Dynamic Programming (DP) is essentially the process of formulating the task in terms of a set of smaller tasks. The use of DP is generally more efficient and effective than the naive methods because most of the time there is a degree of redundancy in the smaller tasks, a redundancy that is removed when DP is used. There are two types of DP approaches: bottom-up and top-down. The bottom-up method basically stores the results of the subtasks, to be used later by other subtasks. The top-down approach, on the other hand, formulates the main tasks as a set of recursive smaller computations. The Viterbi Search algorithm takes advantage of the top-down method of DP as it basically finds the most likely hidden state sequence by recursively finding the most likely hidden state for each observation in the time line separately. The search space of the Viterbi Search algorithm is the set of all possible combinations of hidden states for a given observation sequence having length N (in the case of phoneme recognition N is the number of frames in the speech signal and the observation sequence is the sequence of 25-D feature vectors for each time frame). The search space of the Viterbi Search algorithm is, of course, constrained by the possible initial and final states of the HMM. An example of the space of possible solutions and the solution "path" is demonstrated in Figure 3.3.

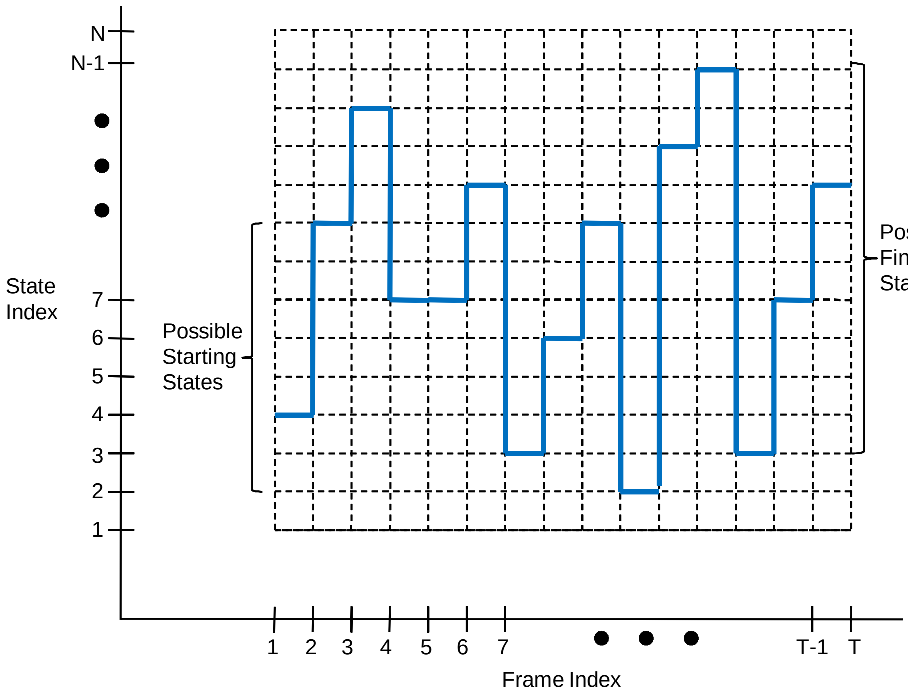

Figure 3.3: A sample solution space of Viterbi paths and a sample solution of the Viterbi Search algorithm.

As we see from Figure 3.3, the possible solutions for the hidden state sequence having maximum likelihood is constrained by the set of possible starting states and a set of possible final states. For the ergodic HMM this constraint does not exist since all the states are valid starting and final states. However, for the left-to-right HMM there is only one valid starting hidden state and one valid final hidden state. This constraint aids in the speed of the Viterbi Search algorithm as will be apparent from the next few sections. The application of the Viterbi Search algorithm to the Segmental HMM is different from that of the regular HMM since the Segmental HMM is a generalization of the regular HMM. In the case of the Segmental HMM, the Viterbi Search algorithm has two goals. The first goal is to find a hidden state sequence that maximizes the likelihood of the observation sequence for a given model. The second goal is to find the hidden state sequence boundaries that maximize the likelihood of the hidden state sequence (since there is a variable number of observations integrated in the emission probability computation of a single hidden state in a Segmental HMM). These two goals are accomplished simultaneously by the same top-to-bottom DP based Viterbi Search algorithm. As a result of the additional second task of the Viterbi Search algorithm for the Segmental HMM the solution space and the solution are generalized versions of those illustrated in the example in Figure 3.3. Figure 3.4 shows an example solution space and an example solution "path" of the Viterbi Search algorithm for the Segmental HMM.

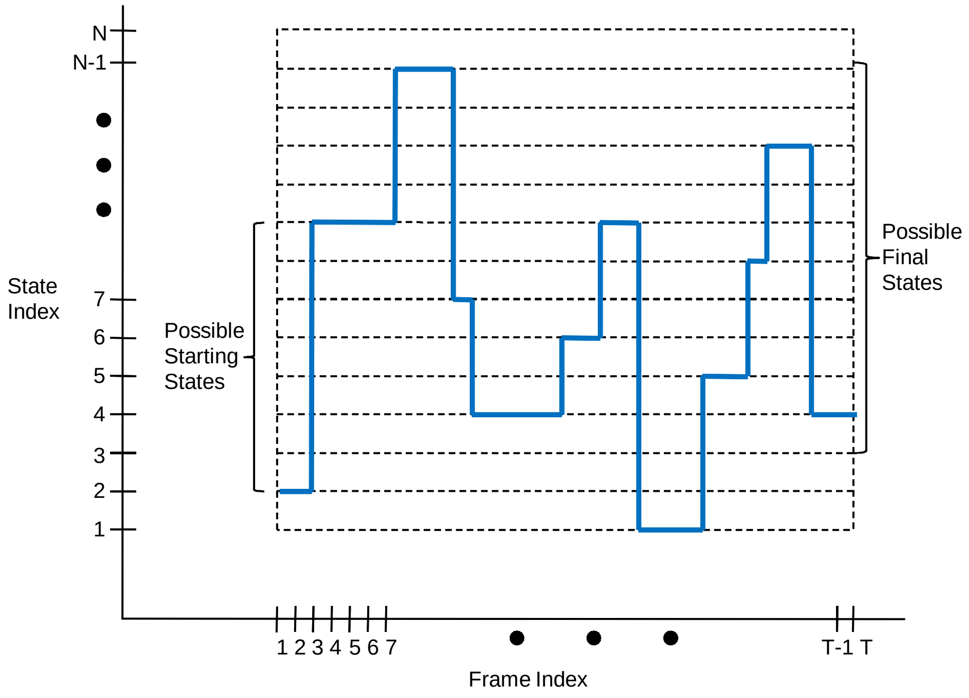

Figure 3.4: A sample solution space and a sample solution of the Segmental Viterbi Search algorithm.

Note the difference between the solution in Figure 3.3 and the solution in Figure 3.4. The duration of each of the hidden states in Figure 3.3 is constant and corresponds to each time frame in the speech signal. However, the duration of each hidden state of the solution in Figure 3.4 is variable as there is a variable number of time frames that corresponds to a single hidden state in a Segmental HMM. In the next two sections of the thesis the theoretical details of the Viterbi Search algorithm for the left-to-right HMM and for the ergodic Segmental HMM are discussed.

3.3.1 Viterbi Search Algorithm for Left-Right HMM

We now discuss the Viterbi Search algorithm for the left-to-right HMM. Let $Q = [q_1, q_2, \ldots, q_N]$ be the sequence of hidden states and $O = [O_1, O_2, \ldots, O_N]$ be the observation sequence. Let the model λ be defined by the parameter set given in (3.9). The likelihood of the hidden state sequence is the probability of the observation sequence and the sequence of hidden states given the model λ (with the assumption that the observations are independent) as shown below:

$$L_p = P(O, Q; \lambda) = \pi_{q_1} \prod_{n=2}^{N} a_{q_n,q_{n-1}} \cdot b_{q_n}^{O_n} \tag{3.15}$$

where $a_{q_n,q_{n-1}}$ are the transition probabilities from state q_{n-1} to state q_n, $b_{q_n}^{O_n}$ is the emission probability from hidden state q_n to observation O_n and π_{q_1} is the initial probability of the first hidden state. For the left-to-right model π_{q_1} has a value of 1 for the first state and 0 for all other states.

Equation (3.15) shows the likelihood of the hidden state sequence and the observation sequence given the model. However, for a large N the likelihood L_p becomes very small, and in many practical applications this leads to numerical instabilities, meaning the approach is then prone to rounding errors. To eliminate the numerical instability problem the likelihood measure in (3.15) is computed in terms of logarithms as shown below

$$L = \log(L_p) = \log\left(P(O, Q; \lambda)\right) = \log(\pi_{q_1}) + \sum_{n=2}^{N} \log(a_{q_n,q_{n-1}}) + \sum_{n=2}^{N} \log(b_{q_n}^{O_n}) \tag{3.16}$$

Because of the fact that the logarithm function $y = \log(x)$ is a monotonically increasing function of x for any value of x in the positive real space, maximizing L_p maximizes L and-conversely-minimizing L_p will minimize L. The goal of the Viterbi Search algorithm (for the left-to-right HMM) is to find an optimum hidden state sequence Q, i.e. one that maximizes the likelihood measure L. Thus

$$Q^* = \underset{Q}{\operatorname{argmax}} \left(\log(\pi_{q_1}) + \sum_{n=2}^{N} \log(a_{q_n,q_{i-1}}) + \sum_{n=2}^{N} \log\left(b_{q_n}^{O_n}\right) \right) \tag{3.17}$$

where $Q^* = [q_1^*, q_2^*, \ldots, q_N^*]$ is the optimum hidden state sequence. Finding Q^* in (3.17) directly is computationally complex and contains many redundant computations. The Viterbi Search algorithm utilizes top-to-bottom DP to reduce the task of finding an optimum hidden state sequence Q^* according to (3.17) to a set of recursive functions. To illustrate this we define a measure

$$\delta_n^i = \max_{q_1,q_2,\cdots,q_{n-1}} \log\left(P(q_1, q_2, \ldots, q_{n-1}, q_n = i, O_1, O_2, \ldots, O_n; \lambda)\right) \tag{3.18}$$

where for each hidden state i at time index n, δ_n^i represents the maximum probability of the hidden state sequence up to frame n and the hidden state i at frame n for the observation sequence $O_1, O_2, \ldots, O_n$ given the model λ. Note that δ_N^i is equivalent to maximizing the criterion L given in (3.16). By induction we have:

$$\delta_n^j = \max_i \left(\delta_{n-1}^i + \log(a_{j,i})\right) + \log\left(b_j^{O_n}\right) \tag{3.19}$$

Equation (3.19) is the essential recursive formulation that constitutes the core of the Viterbi Search algorithm. The Viterbi Search algorithm also uses an indexing variable Ψ_n^i for storing the hidden state sequence at time index n that has the highest likelihood. Now that we have defined all the parameters of the Viterbi Search algorithm, the Viterbi Search algorithm is outlined below.

1. **Initialization:**

$$\delta_1^i = \log(\pi_i) + \log(b_i^{O_1}) \tag{3.20}$$

$$\Psi_1^i = 0 \quad \forall\, i; \tag{3.21}$$

2. **Recursion:**

$$\delta_n^j = \max_{1 \le i \le S} \left(\delta_{n-1}^i + \log(a_{j,i})\right) + \log(b_j^{O_n}) \quad for \quad 2 \le n \le N; \quad 1 \le i \le S \tag{3.22}$$

$$\Psi_n^j = \underset{1\le i\le N}{\operatorname{argmax}} \left(\delta_{n-1}^i + \log(a_{j,i})\right) \quad for \quad 2 \le n \le N; \quad 1 \le i \le S \tag{3.23}$$

3. **Termination:**

$$L^* = \max_{1\le i\le S}(\delta_N^i) \tag{3.24}$$

$$q_T^* = \underset{1\le i\le S}{\operatorname{argmax}}(\delta_N^i) \tag{3.25}$$

4. **Backtracking:**

$$q_n^* = \Psi_{n+1}^{q_{n+1}^*} \quad for \quad n = N-1, N-2, \ldots, 1 \tag{3.26}$$

In (3.26) the sequence $[q_1^*, q_2^*, \ldots, q_N^*]$ represents the single most likely hidden state sequence and L^* in (3.24) represents the likelihood of the most likely hidden state sequence. The next section discusses the generalization of the Viterbi Search algorithm for Segmental HMMs.

3.3.2 Viterbi Search Algorithm for Segmental Ergodic HMM

In the previous section the Viterbi Search algorithm for the left-to-right HMM was outlined. In this section we present a generalization of the Viterbi Search algorithm for use with the segmental ergodic HMM and thus is called the Segmental Viterbi Search algorithm. The Segmental Viterbi Search algorithm simultaneously finds the most likely sequence of the hidden states and the most likely segmentation of the observation sequence to the hidden states, conditioned on several constraints. To illustrate the operation of the Segmental Viterbi Search algorithm we first define the variables to be optimized and used in the algorithm. Let the observation set be $O = [o_1, o_2, \ldots, o_N]$, a sequence of hidden states is denoted by $Q = [q_1, q_2, \ldots, q_M]$, and the corresponding sequence of most likely hidden states by $Q^* = [q_1^*, q_2^*, \ldots, q_M^*]$. Note that the number of hidden states and the most likely hidden states are different from the number of observations. This is because of the fact that - for a Segmental HMM - there may be more than one observation that corresponds to a particular hidden state.

The start and endpoints of each of the hidden states is represented by an array of start-stop pairs $\Gamma = [< F_L^i, F_H^i >_1, < F_L^i, F_H^i >_2, \ldots, < F_L^i, F_H^i >_M]$ where F_L^i is the index of the first (low) observation that corresponds to the state i and F_H^i is the index of the last (high) observation that corresponds to the state i. The start-stop pair $< F_L^i, F_H^i >_n$ corresponds to the first and last observation indices of the hidden state q_n in the hidden state sequence. Similarly, $\Gamma^* = [< F_L^i, F_H^i >_1^*, < F_L^i, F_H^i >_2^*, \ldots, < F_L^i, F_H^i >_M^*]$ represents the start-stop pairs for hidden states that belong to the most likely hidden state sequence. In the phoneme recognition system, each of the hidden states in the Segmental HMM represents a phoneme state, and as such, is bound to a length constraint having a minimum length and a maximum length indicated by $q_{i,\min}$ and $q_{i,\max}$ respectively for hidden state i. The length of a phoneme state is modeled as a Gaussian distribution and the quantities $q_{i,\min}$ and $q_{i,\max}$ are determined by taking d_1 and d_2 standard deviations smaller and bigger than the mean of the phoneme state length respectively. The phoneme state length statistics are learned during the training of the Segmental HMM. The possible set of observations that can be spanned by each of the hidden states is subject to two constraints and depends on the stop observation index of the previous hidden state and the minimum and maximum length constraints of the hidden state. Together these constraints provide a range for the possible starting points of the hidden state. The first constraint determines the lower index of the possible starting observation indices of the hidden state by allowing the first observation to be up to ρ observations prior to the last observation of the previous hidden state. Likewise the upper index of the possible observation points is allowed to be up to ρ points after the last observation index of the previous hidden state. Each of the emission likelihoods and the start and endpoints of each hidden state are determined from the possible range of observations that can belong to the hidden state, subject to the two constraints. The range of valid observations that can belong to a hidden state (based on the length constraint and the starting observation constraint) is illustrated in Figure 3.5.

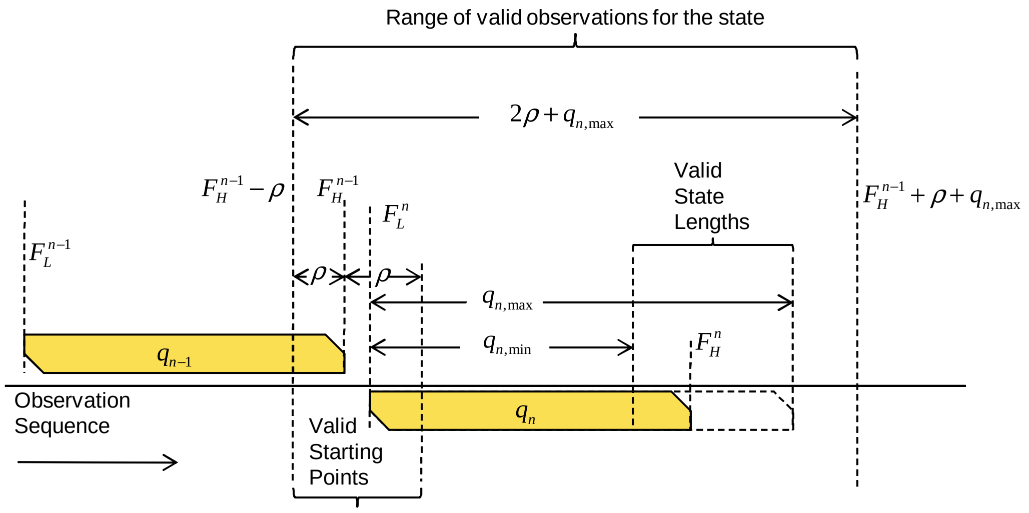

Figure 3.5: Diagram outlining a set of observations that can be associated with a hidden state in Segmental HMM.

As illustrated in Figure 3.5 the total number of valid observations that is needed for the determination of start and end observation indices of the hidden state q_n is $2\rho + q_{n,\max}$. The lower limit of the valid observation sequence array for the hidden state q_n is given by $F_H^{n-1} - \rho$. Likewise, the upper limit of the valid observation sequence array for the hidden state q_n is given by $F_H^{n-1} + \rho + q_{n,\max}$ as shown in Figure 3.5. Two functions - defined by the wrappers $g_i(L, H)$ and $h_i(L, H)$ - are used in the segmental Viterbi Search algorithm for determining the emission probability of a given state i, and the first and the last indices of the hidden state i respectively. Both of these functions take the lower and upper observation index limits of the array of valid observations for state q_n as their input (L and H respectively). The function $g_i(L, H)$ returns the emission probability of the state i. Furthermore, the function $h_i(L, H)$ returns the start-stop pair $< F_L^i, F_H^i >$ of the state i. Similar to the Viterbi Search algorithm for the left-to-right HMM the Segmental Viterbi Search algorithm relies on the computational advantages of top-down dynamic programming. As such, we need to define the following quantity, that is used in the recursive process of the DP:

$$\delta_n^i = \max_{q_1, q_2, \ldots, q_{n-1}} \log\left(P(q_1, q_2, \ldots, q_{n-1}, q_n = i, O_1, O_2, \ldots, O_{F_H^n}; \lambda)\right) \tag{3.27}$$

where δ_n^i represents the maximum of the joint likelihood of all the possible hidden state sequence combinations $q_1, q_2, \ldots, q_{n-1}$ and their corresponding possible configurations of the observation arrays ($O_{F_L^n}, O_{F_L^n+1}, \ldots, O_{F_H^n}$ for each hidden state q_n) and the likelihood of hidden state q_n being i. By induction we have:

$$\delta_n^j = \max_i \left(\delta_{n-1}^i + \log(a_{j,i})\right) + \log\left(g_j(F_H^{n-1} - \alpha, F_H^{n-1} + q_{j,\max} + \beta)\right) \tag{3.28}$$

Equation (3.28) is used in the recursive process of the Segmental Viterbi Search algorithm as the criterion on which the selection of the hidden state and its start-stop points is based. The function $g_i(L, H)$ is used here as the emission likelihood of the hidden state, given the range of valid observations $O_{F_H^{n-1}-\rho}, O_{F_H^{n-1}-\rho+1}, \ldots, O_{F_H^{n-1}+q_{i,\max}+\rho}$ that can be associated with the hidden state i. There are two additional variables, used for indexing and record keeping. The first one, Ψ_n^i, stores the label associated with the hidden state i that maximizes the right hand side of (3.28) for all the possible S hidden states. Finally, $\Omega_n^i =< \Omega_L^i(n), \Omega_H^i(n) >$ represents the start and stop observation indices that correspond to Ψ_n^i. Now that all the variables used in the recursive formulation of the Segmental Viterbi Search algorithm have been defined, the Segmental Viterbi Search algorithm is summarized.

1. **Initialization:**

$$\delta_1^i = \log(\pi_i) + g_i(0, q_{i,\max} + \beta) \quad \forall\, i; \quad (q_{i,\max} + \rho) \leq N \tag{3.29}$$

$$\Psi_1^i = 0 \quad \forall\, i; \quad (q_{i,\max} + \rho) \leq N \tag{3.30}$$

$$\Omega_1^i = h_i(0, q_{i,\max} + \rho) \quad \forall\, i; \quad (q_{i,\max} + \rho) \leq N \tag{3.31}$$

2. **Recursion:**

$$\delta_n^j = \max_{\forall\, i:\ (\Omega_H^i(n-1)+q_{i,\max}+\rho)\leq N} \left(\delta_{n-1}^i + \log(a_{j,i})\right) + g_j(\Omega_H^j(n-1) - \rho, \Omega_H^j(n-1) + q_{j,\max} + \rho)$$
$$\forall\, n, j:\ (\Omega_H^j(n-1) + q_{j,\max} + \rho) \leq N \tag{3.32}$$

$$\Psi_n^j = \operatorname*{argmax}_{\forall\, i:\ (\Omega_H^i(n-1)+q_{i,\max}+\rho)\leq N} \left(\delta_{n-1}^i + \log(a_{j,i})\right) \quad \forall\, n, j:\ (\Omega_H^j(n-1) + q_{j,\max} + \rho) \leq N \tag{3.33}$$

$$\Omega_n^j = h_j(\Omega_H^i(n-1) - \rho, \Omega_H^i(n-1) + q_{j,\max} + \rho) \quad \forall\, n, j : \; (\Omega_H^j(n-1) + q_{j,\max} + \rho) \leq N \tag{3.34}$$

3. **Termination:**

$$L^* = \max_{\forall\, i,n:\; (\Omega_H^i(n-1)+q_{i,\max}+\rho)\leq N} (\frac{\delta_n^i}{n}) \tag{3.35}$$

$$\Upsilon^* = \operatorname*{argmax}_{n:\forall\, i:\; (\Omega_H^i(n-1)+q_{i,\max}+\rho)\leq N} (\frac{\delta_n^i}{n}) \tag{3.36}$$

$$q_{\Upsilon^*}^* = \operatorname*{argmax}_{i:\forall\, n:\; (\Omega_H^i(n-1)+q_{i,\max}+\rho)\leq N} (\frac{\delta_n^i}{n}) \tag{3.37}$$

4. **Backtracking:**

$$q_n^* = \Psi_{n+1}^{q_{n+1}^*} \; for \quad 1 \leq n \leq \Upsilon^* - 1 \tag{3.38}$$

$$\Gamma_n^* = \Omega_n^{q_n^*} \; for \quad 1 \leq n \leq \Upsilon^* - 1 \tag{3.39}$$

where L^* represents the likelihood of the combination of the most likely hidden state sequence and the most likely distribution of the observations into the hidden states. The number of hidden states in the hidden state sequence varies and depends on the number of observations that was selected during the computation of the emission probabilities of each of the hidden states. Because of the variability in the lengths of the observation array assigned to each of the hidden states, and the constraints imposed on the range of observations that can be assigned to the hidden states, the most optimum hidden state sequence can consist of any number of hidden states S, where S ranges from 1 to the length of the largest hidden state sequence that can fit in the observation sequence (thus S has an upper limit of N, which corresponds to the case of a regular HMM having assigned precisely one observation to each hidden state). The likelihoods of the hidden state sequences having different lengths can not be compared directly to each other because a hidden state sequence with bigger length is bound to have less likelihood than a hidden state with a smaller length. This is because the joint probability of the states in the hidden state sequence with larger size has larger dimensionality than the joint probability of the states in the hidden state sequence that has smaller length. For this reason (3.35),(3.36), and (3.37) normalize each hidden state likelihood in δ_n^i by the length of the hidden state sequence n. This normalization is valid because each observation in a Viterbi path adds an equal amount of likelihood to the Viterbi path likelihood. Thus the last hidden state index and its corresponding index of the most likely hidden state is found by maximizing $\frac{\delta_n^i}{n}$ in (3.35) and is given by $q^*_{\Upsilon^*}$ and Υ^*. In (3.36) Υ^* is the index of the last hidden state of the chosen hidden state sequence and $q^*_{\Upsilon^*}$ in (3.37) is the corresponding last hidden state. The backtracking step in the Segmental Viterbi Search algorithm is done in the same manner as in the regular Viterbi Search algorithm, but starting with the last hidden state Υ^* in the hidden state sequence.

Maximizing the per-state likelihoods $\frac{\delta_n^i}{n}$ is not the most practical method for finding the last state of the optimal hidden state sequence since the possibility of having the most likely Viterbi path towards the beginning of the observation sequence is more likely than having it towards the end. This method is referred to as the Scaled Viterbi Likelihood method of finding the optimal hidden state sequence in this thesis. The simulations and analysis in the next chapter show that only a small percentage of the observations are actually classified to hidden states when the Scaled Viterbi Likelihood method is used. A more robust way of finding the last hidden state of the most optimum hidden state sequence is to choose the hidden state in Ψ_n^i that, after backtracking from state $q_n = i$, contains the largest number of hidden states maximizing δ_{n1}^i over all the hidden states i, where $1 \leq n1 \leq n$. Thus in the actual implementation of the segmental Viterbi Search algorithm, during each computation of the δ_n^i the algorithm also performs the backtracking step from state $q_n = i$ and stores the number of states $q_{n1} = i$ that maximized δ_{n1}^i over all the hidden states i. Furthermore, during the termination step, the segmental Viterbi Search algorithm chooses the state $q_n = i$ that contains the largest number of hidden states maximizing δ_{n1}^i over all the hidden states i. This method of finding the optimum hidden state sequence is referred to as the Max Viterbi Likelihood method in this thesis.

So far we have presented the application of the Viterbi Search algorithm to the Segmental HMM. We now discuss the implementation of the functions $g_i(L, H)$ and $h_i(L, H)$. The function $g_i(L, H)$ determines the maximum likelihood of the observation arrays ranging from size $q_{i,\max}$ to size $q_{i,\max}$, starting at the valid starting indices and being assigned to hidden state i. As mentioned before, these observation arrays are part of the observation segment taken from the observation sequence having lower and upper bounds L and H respectively. Let the associated left-to-right HMM model of each of the hidden phoneme states be λ_i. The function $g_i(L,H)$ basically computes the likelihood of the most likely configuration of the observation array by sequentially running the Viterbi Search algorithm for all possible configurations of the observation set (within the L and H bounds), given the observation array and the HMM model λ_i. The likelihood of a particular configuration of the observation set is computed by finding the likelihood of the most likely hidden state sequence (given the model λ_i), by means of the regular Viterbi Search algorithm. Thus the function goes through each possible starting point l where $(F_H^{n-1} - \rho) \leq l \leq F_H^{n-1} + \rho)$ for the state $q_n = i$ and computes a set of likelihoods of the most likely hidden state sequence for the model λ_i. Thus for each value of l the function performs $q_{i,\max} - q_{i,\min}$ Viterbi Search operations. For each of these Viterbi Search operations, the selected observation sequence is $O_k = [O_l, O_{l+1}, \ldots, O_{l+q_{i,min}+k}]$ where $0 \leq k \leq (q_{n,\max} - q_{n,\min})$. Let L_k^l denote the normalized likelihood (normalized by the length of the chosen observation sequence) of the most likely hidden state sequence for which the observation sequence starts at index $F_H^{n-1} - \rho + l$ and stops at index $F_H^{n-1} + q_{n,\min} + k$. The Viterbi decoding process having the largest likelihood for the most likely hidden state sequence is returned as the emission likelihood. Thus the function $g_i(L,H)$ returns

$$g_i(L,H) = \underset{k,l}{\operatorname{argmax}}(L_k^l) \tag{3.40}$$

Let I_1 and I_2 respectively be the starting and ending observation indexes of the observation array configuration yielding the largest likelihood. The function $h_i(L,H)$ records I_1 as the starting point of the hidden state q_n given by:

$$F_L^n = F_H^{n-1} - \rho + \underset{k,l}{\operatorname{argmax}}(L_k^l) \tag{3.41}$$

The stop observation index of the hidden state is found by choosing the first observation after I_1 in the observation sequence for which the change in the Viterbi likelihood L_k^l exceeds a threshold τ. Thus the stop observation index is found to be:

$$F_H^n = F_L^n + q_{i,\min} + \min_k(|L_k^{F_L^n} - L_{k-1}^{F_L^n}| > \tau) \quad for\ 2 \leq k \leq (q_{i,\max} - q_{i,\min}) \tag{3.42}$$

and finally, the function $h_i(L, H)$ returns the pair $< F_L^n, F_H^n >$.

The algorithms used for the recognition of the phonemes has been presented. The next section of the thesis discusses the Baum-Welch algorithm, used in the estimation of the left-to-right HMM Parameters.

3.4 Baum-Welch Re-estimation

The Baum-Welch algorithm is a special case of the Expectation Maximization (EM) algorithm that is used for the estimation of the HMM parameters. Thus the task of the Baum-Welch algorithm is to adjust the parameters of the HMM model (given an arbitrary selection of the initial parameters) to maximize the likelihood of the observation sequences. In the phoneme recognition system the Baum-Welch re-estimation algorithm is applied during the training phase for the estimation of the left-to-right HMM model parameters for each phoneme state. The left-to-right HMM model parameters, in turn, add distinguishing characteristics to each of the phoneme states and are used in the recognition phase of the phoneme recognition system. The estimated model parameters $\lambda_i = (A, B, \pi)$ correspond to each of the hidden phoneme states that are used in the segmental Viterbi Search algorithm presented in the previous section. These left-to-right HMM parameters are used during the determination of the emission probabilities during the segmental Viterbi search. In other words, after the training of the left-to-right HMM parameters (by means of the Baum-Welch algorithm) for each hidden phoneme state, the phoneme recognition system uses those HMM parameters to determine the phoneme state that an observation sequence most likely belongs to. A detailed description of the application of model parameters to the segmentation and recognition of the phoneme states was presented in Section 3.3.2 of this chapter (see the discussion of the implementation of the two functions $g_i(L, H)$ and $h_i(L, H)$). We now illustrate the training and estimation phase of the left-to-right HMM parameters (given the observation set pertaining to the HMM) using the Baum-Welch algorithm. The Baum-Welch re-estimation algorithm is responsible for estimating four different sets of parameters: the state transition matrix A (bound to the state transition constraints of the left-to-right HMM), the weights, and the means and variances of the hidden state Gaussian Mixture Models (used for the computation of the emission probabilities during the Viterbi search). The initial state probability vector π (representing the initial weights of the hidden states) is assumed to have one for the starting state and zeroes for the rest of the states. The Baum-Welch re-estimation algorithm is an iterative two step algorithm. The two steps in each of the iterations of the Baum-Welch algorithm are the Expectation step and the Maximization step. Both of these steps are presented in detail later in this chapter (Subsections 3.4.3 and 3.4.4). Since the Baum-Welch re-estimation algorithm is a particular form of the Expectation Maximization algorithm, the likelihood of the observation set is guaranteed to converge (proof of which can be found elsewhere [22]). However, being a particular case of the EM algorithm, the Baum-Welch algorithm is prone to get trapped at a local maximum. Thus, "good" initialization is crucial for the re-estimation performance of the Baum-Welch Algorithm.

This section of the thesis is organized as follows. First we discuss the algorithms that are used in this thesis for initializing the Baum-Welch algorithm. These are the K-Means clustering and the corresponding Expectation Maximization algorithm that is applied to the clustering task. Then we present the forward-backward algorithm which is used extensively in the Baum-Welch algorithm. Finally we present the Expectation and the Maximization steps of the Baum-Welch algorithm.

3.4.1 Initialization of the HMM Model

The EM algorithm generally provides good convergence under any initial guesses of the HMM model parameters, however it is possible for the Baum-Welch algorithm to converge to local maxima. As such, we use several techniques and algorithms for introducing a random initialization scheme that provides good initial parameters for the Baum-Welch re-estimation.

The HMM parameters that are initialized prior to the start of the Baum-Welch re-estimation process are: the state transition matrix A, the weights c_{ik}, the means μ_{ik}, and the covariances Σ_{ik} of the hidden state Gaussian Mixture Models for a hidden state i and mixture k. To initialize these parameters we first assume that the hidden state transitions have the same probability. Thus for the case of an S-state left-to-right HMM the elements a_{ij} in the transition probability are: 1 for $i = j = S$, 0.5 for $i = j$ and $i = j + 1$ and 0 for all other i and j. The initial transition probability matrix is illustrated below:

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0 & 0 & 0 & \cdots & 0 \\ 0.5 & 0.5 & 0 & 0 & \cdots & 0 \\ 0 & 0.5 & 0.5 & 0 & \cdots & 0 \\ 0 & 0 & 0.5 & 0.5 & \cdots & 0 \\ 0 & 0 & 0 & 0.5 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0.5 & 1 \end{bmatrix} \tag{3.43}$$

We then assume that occupancy of each of the states in the observation sequences are equiprobable. To incorporate this assumption into the initialization process of the HMM parameter set we equally and uniformly distribute the entire set of observation sequences, comprising the total array of observation points $O = [o_1, o_2, \ldots, o_N]$, into S different arrays of observations $O_i = [o_1, o_2, \ldots, o_{P^i}]$ where P^i is the size of each observation partition i, $1 \leq i \leq S$ and $\sum P^i = N$. Each of the observation arrays is used for the initialization of the GMM parameters (the GMM weights, means, and covariances). Each of the observation arrays O_i (for each state i) is clustered into K different clusters. The initial GMM parameters of each state i are the weights, the means, and the covariance matrixes, computed from these clusters. Two types of clustering are performed for the generation of the GMM parameters: hard clustering and soft clustering. The hard clustering is responsible for the clustering of the data by minimizing the within cluster sum of squared errors (WCSSE) computed from the Euclidian distances between the observation data-points and their corresponding cluster centers. On the other hand, the soft clustering is responsible for clustering the data by maximizing the likelihood of the data points given the mixture parameters μ_{ik}, Σ_{ik}, and c_{ik}. The clustering of the data for the initialization of GMM parameters of the HMM is done by first applying a hard clustering algorithm, K-Means, and then a soft clustering algorithm, EM Clustering, in a sequential order. The K-Means clustering is applied for finding the cluster centers. With the cluster centers known, EM clustering is applied for the estimation of the GMM parameters. The hard clustering is performed first to prevent errors from rounding during the clustering operation. After hard clustering, a soft clustering method is applied for more accurate clustering of the data coming from a GMM. The K-Means and the EM Clustering algorithms are explained in greater detail in the three subsections below.

After the determination of the initial selection of the GMM parameters we randomize these parameters R times. During each randomization procedure, the GMM parameters (weights, means, and covariances) are randomized as follows:

$$\widehat{c_{ik}} = c_{ik} \tag{3.44}$$

$$\widehat{\mu_{ik}} = \frac{\mu_{ik}}{2} + r\mu_{ik} \tag{3.45}$$

$$\widehat{\Sigma_{ik}} = \frac{\Sigma_{ik}}{2} + r\Sigma_{ik} \tag{3.46}$$

where $\widehat{c_{ik}}$, $\widehat{\mu_{ik}}$, and $\widehat{\Sigma_{ik}}$ are the weights, means, and covariances used for each of the R random initializations and r is a random variable uniformly distributed between 0 and 1. The Baum-Welch algorithm is rerun for each of these R random initializations and the model parameter set yielding the highest likelihood of the observation set is chosen as the final set of HMM model parameters.

Initializing the K-Means Algorithm

The initialization of the K-Means algorithm plays a key role in the performance of the hard clustering, because the K-Means algorithm is very similar to the Expectation Maximization algorithm, and is prone to the possibility of being trapped in a local minimum. Thus a good initialization procedure will give a significant advantage to the performance of the K-Means algorithm. The paper by Douglas Steinley [23] provides an excellent review of the different initialization techniques used for the initialization of the K-means algorithm and their corresponding tradeoffs. The algorithm used in this thesis for the initialization of the cluster centers for the K-means algorithm is chosen to be a slightly modified version of the MinMax, or Farthest First (FF) algorithm [24]. The Farthest First algorithm effectively initializes the K-Means algorithm by choosing those data points as the means that cause the cluster centers to be as far from each other as possible. This initialization choice ensures that the centers of the clusters will come close to the global data mean during the iterations of the K-Means clustering, and will maximize the between-cluster spread of the cluster centers.

The FF algorithm operates as follows. We first compute the global mean. After the global mean has been computed the FF algorithm searches for the data point having the greatest Euclidian distance from the global mean and assigns it as the first center. After the first center has been chosen the FF algorithm chooses the next $K-1$ centers by searching for data points in the observation array that are furthest from all the previously chosen centers. The FF procedure is formally stated below:

1. **Computing the Global Mean:**

$$\overline{\mu}_i = \frac{1}{P_i}\sum_{n=0}^{P_i} o_n^i \tag{3.47}$$

where $\overline{\mu}_i$ is the global mean of the observation array $O_i = [o_1, o_2, \ldots, o_{P^i}]$ and P_i is the size of the observation array O_i corresponding to the hidden state i for which the GMM parameters are being estimated.

2. **Finding the First Center:**

$$\mu_1 = \underset{O_i}{\operatorname{argmax}} \sum_{j=1}^{D} (o_n^i(j) - \overline{\mu}_i(j))^2 \tag{3.48}$$

where μ_1 is the first center and is one of the data points maximizing (3.48) and D is the dimensionality of the observation set. The quantity $o_n^i(j)$ in (3.48) represents the dimension j of the observation point at time index n generated from the hidden state i.

3. **Finding the $K-1$ Centers:**

$$\mu_k = \underset{O_i}{\operatorname{argmax}} \min_{l} \sum_{j=1}^{D} (o_n^i(j) - \mu_l(j))^2 \quad for\ 1 \leq l \leq k-1 \tag{3.49}$$

where μ_k is the k^{th} center and $2 \leq k \leq K$.

After the initialization of the K-Means algorithm we proceed by presenting the subsequent steps of the K-Means algorithm in the next subsection.

K-Means Algorithm

The purpose of the K-Means algorithm in this thesis is to partition an observation data set having size P_i into K observation sets such that the within cluster sum of squared error is minimized (WCSSE). Let $\Theta = [\Theta_1, \Theta_2, \ldots, \Theta_K]$ denote the partitioned observation sets. The goal of the K-Means algorithm is to partition the observation data set that minimizes the within cluster sum of squared error:

$$WCSSE = \sum_{k=1}^{K} \sum_{o_n \in \Theta_k} (o_n - \mu_k)^2 \tag{3.50}$$

Minimizing $WCSSE$ in (3.50) directly is an NP-Hard problem and thus not feasible for any practical application. As such, the K-Means algorithm utilizes a technique similar to the Expectation Maximization algorithm to find a partition set Θ and the corresponding cluster means μ_i that minimizes the $WCSSE$ in (3.50). The K-Means algorithm is an iterative two step process. During the first step of the iteration each data point o_n in the observation set is categorized into one of the K different clusters. After classifying each data point in the data set to the cluster with its mean having the smallest Euclidian distance to the data point, the means of all clusters are updated. The adjustment of the means of the clusters is updated by computing the mean of each partition Θ_i. This process is repeated iteratively until either the change in the $WCSSE$ value is smaller than a chosen threshold, or until the maximum permissable number of iterations is reached. The K-Means algorithm is formally stated below:

1. **Initialization:**

   Initialize the centers of the clusters μ_i using any initialization method (in this case the FF algorithm).

2. **Assignment of data to cluster partitions:**

   Assign the data points to the cluster partition according to:

$$\Theta_k^m(n) = o_n : \sum_{j=1}^{D}(o_n(j) - \mu_k^m(j))^2 \leq \sum_{j=1}^{D}(o_n(j) - \mu_{k^*}^m(j))^2 \; for \; 1 \leq k^* \leq K \; and \; k^* \neq k \tag{3.51}$$

$$WCSSE^m = \sum_{k=1}^{K} \sum_{o_n \in \Theta_k^m} (o_n - \mu_k^m)^2 \tag{3.52}$$

   where D is the dimensionality of the data and the cluster centers and $\Theta_k^m(n)$ represents the data point n in the k^{th} partition during the iteration m. The size of each data partition Θ_k^m is denoted by P_k^m.

3. **Updating the means:**

$$\mu_k^{m+1} = \frac{1}{P_k^m} \sum_{n=1}^{P_k^m} \Theta_k^m(n) \tag{3.53}$$

$$WCSSE^{m+1} = \sum_{k=1}^{K} \sum_{o_n \in \Theta_k^{m+1}} (o_n - \mu_k^{m+1})^2 \tag{3.54}$$

   Set $m = m + 1$ and then go back to step 2 if $|WCSSE^{m+1} - WCSSE^m| \geq \eta$ and $m \leq \phi$, where η is the convergence threshold and ϕ is the maximum number of iterations permitted for the K-Means algorithm; otherwise stop.

We have presented the K-Means algorithm in this subsection. The next subsection discusses the EM Clustering algorithm.

EM Clustering Algorithm

The EM Clustering algorithm is used in this thesis for the computation of the GMM parameters associated with each hidden state in the left-to-right HMM. Similar to K-Means clustering, EM Clustering is an iterative two-step process having the Expectation step and the Maximization step. However, unlike K-Means clustering, the cluster covariances and weights are updated based on the likelihood of the data points being generated from a particular mixture. The EM clustering algorithm is a soft clustering algorithm since the membership of the observation data points to a mixture is defined by a likelihood measure (as opposed to a direct membership classification as is the case in the K-Means algorithm). Hence in the K-Means algorithm a data point is a member of a single cluster only, while in EM the extent of membership in a cluster is a degree of membership. Thus the goal of EM clustering is to choose the GMM mixture parameters that maximize the log likelihood of the observation set given by:

$$L = \sum_{k=1}^{K} \sum_{n=1}^{P_k} (\log(c_k) + \log(P(o_n; \theta_k))) \tag{3.55}$$

where θ_k is the mixture parameter set which consists of the mixture mean μ_k and the mixture covariance matrix Σ_k, and the mixture weight c_k. Similar to the K-Means algorithm, the GMM parameters have to be initialized for the EM Clustering algorithm. In this thesis, the initialization of the GMM parameters is done by the K-Means algorithm. The centers of the mixtures are computed by the K-Means algorithm, while the covariance matrices and the mixture weights are obtained from each individual partition derived by the K-Means algorithm. During the Expectation step of the iteration, the EM Clustering algorithm computes membership probabilities $P(\psi_n|o_n)$ where ψ_n denotes a random variable representing the mixture that the data point o_n belongs to. The maximization step of the EM clustering iteration uses the updated membership probabilities to compute the weighted mean, weighted covariance, and the cluster weights. This two step process is iteratively repeated until either the change in the likelihood measure L given in (3.55) is below the threshold η_{EM} or the number of permissable iterations ϕ_{EM} is exceeded. The EM clustering algorithm is formally presented below while the full derivation of the EM clustering algorithm is shown in the Appendix:

1. **Initialization:**

Initialize the GMM parameters: the cluster weights, means, and covariance matrixes - c_k, μ_k, and Σ_k respectively - by the K-Means algorithm.

2. **Expectation Step:**

$$P(\psi_n = k|o_n)^m = \frac{c_k^m N(\mu_k^m, \Sigma_k^m, o_n)}{\sum_{i=1}^{K} c_i^m N(\mu_i^m, \Sigma_i^m, o_n)} \tag{3.56}$$

$$L^m = \sum_{k=1}^{K} \sum_{n=1}^{P_k} (\log(c_k^m) + \log(P(o_n; \theta_k^m))) \tag{3.57}$$

3. **Maximization Step:**

$$\mu_k^{m+1} = \frac{\sum_{n=1}^{P_i} P(\psi_n = k|o_n)^m o_n}{\sum_{n=1}^{P_i} P(\psi_n = k|o_n)^m} \tag{3.58}$$

$$\Sigma_k^{m+1} = \frac{\sum_{n=1}^{P_i} P(\psi_n = k|o_n)^m (o_n - \mu_k)(o_n - \mu_k)^T}{\sum_{n=1}^{P_i} P(\psi_n = k|o_n)^m} \tag{3.59}$$

$$c_k^{m+1} = \frac{1}{P_i} \sum_{n=1}^{P_i} P(\psi_n = k|o_n)^m \tag{3.60}$$

$$L^{m+1} = \sum_{k=1}^{K} \sum_{n=1}^{P_k} \left(\log(c_k^{m+1}) + \log(P(o_n; \theta_k^{m+1}))\right) \tag{3.61}$$

Set $m = m + 1$ then go to step 2 if $|L^{m+1} - L^m| \geq \eta_{EM}$ and $n \leq \phi_{EM}$, where η_{EM} is the convergence threshold and ϕ_{EM} is the maximum number of iterations permitted for the EM Clustering algorithm; otherwise stop.

3.4.2 Forward-Backward Algorithms

In the previous section we have discussed the initialization of the Baum-Welch algorithm. We now proceed with the presentation of the Baum-Welch Expectation Maximization algorithm. The Forward-Backward procedure is an efficient method for computing the probability of the observation sequence given the left-to-right HMM model parameters. Consider the observation sequence $O = [O_1, O_2, \dots, O_N]$ and the corresponding hidden state sequence $Q = [q_1, q_2, \dots, q_N]$, each of which is labeled by one of the possible S hidden states. The parameter set of the HMM is given by $\lambda = (A, B, \pi)$ where A is the state transition matrix, B corresponds to the emission probabilities, and π is the initial hidden state weight vector. The probability of the observation sequence O given a sequence of hidden states Q and the assumption of independence between the observations, is given by:

$$P(O|Q;\lambda) = \prod_{n=1}^{N} P(O_n|q_n;\lambda) = \prod_{n=1}^{N} b_{q_n}^{O_n} \tag{3.62}$$

where $b_{q_n}^{O_n}$ is as defined in (3.12) for a continuous left-to-right HMM. Similarly, the probability of a hidden state sequence is given by:

$$P(Q;\lambda) = \pi_{q_1} \prod_{n=2}^{N} a_{q_n,q_{n-1}} \tag{3.63}$$

The joint probability of the observations and the hidden states is given by the product of the marginal probability of the states and the conditional probability of the observation sequence and is represented as:

$$P(O,Q;\lambda) = P(O|Q;\lambda)P(Q;\lambda) = b_{q_1}^{O_1}\pi_{q_1}\left(\prod_{n=2}^{N} b_{q_n}^{O_n} a_{q_n,q_{n-1}}\right) \tag{3.64}$$

To obtain the marginal probability of the observation sequence, we sum the joint probability over all the possible hidden state sequences. Thus the marginal probability $P(O;\lambda)$ is given by:

$$P(O;\lambda) = \sum_{q_1,q_2\ldots,q_N} P(O|Q;\lambda)P(Q;\lambda) = \sum_{q_1,q_2\ldots,q_N} b_{q_1}^{O_1}\pi_{q_1}\left(\prod_{n=2}^{N} b_{q_n}^{O_n} a_{q_n,q_{n-1}}\right) \tag{3.65}$$

This process is computationally very expensive and does not take advantage of the Markov assumption. Therefore the direct computation of $P(O;\lambda)$ in (3.65) is also very inefficient. The Forward-Backward algorithm provides an easy and efficient method for computing the probability of the observation sequence. The Forward-Backward procedure takes advantage of Dynamic Programming to formulate a recursive algorithm for the computation of the observation likelihood. The detailed implementations of the forward and backward probabilities - and how they contribute to the computation of the observation likelihood - are given in the next two subsections respectively.

Forward Probabilities

The forward probabilities are defined as the joint probability of the partial observation sequence $O_1, O_2, \dots, O_n$ and the event of the hidden state q_n at time n being i, given the HMM model parameters. Thus the forward probabilities α_n^i are given by:

$$\alpha_n^i = P(O_1, O_2, \dots, O_n, q_n = i; \lambda) \tag{3.66}$$

Note that this probability measure is similar to δ_n^i in (3.27) and (3.18) with the difference that the δ_n^i maximizes the hidden state sequence probability. Thus, as was seen in the description of the Viterbi Search algorithms, the forward probabilities constitute the core formulation of the DP approach to the task of computing the likelihood of the observation sequence. The forward probabilities are computed as follows:

1. **Initialization:**
$$\alpha_1^i = \pi_i b_1^i \quad for\ 1 \le i \le S \tag{3.67}$$

2. **Induction Step:**
$$\alpha_{n+1}^i = \sum_{j=1}^{S} (a_{ij}\alpha_n^j) b_{n+1}^i \quad for\ 1 \le n \le N-1; \quad for\ 1 \le i \le S \tag{3.68}$$

3. **Termination step:**
$$P(O; \lambda) = \sum_{i=1}^{S} \alpha_N^i \tag{3.69}$$

Backward Probabilities

The backward probabilities are computed in a similar manner as the forward probabilities. The difference between the computation of the forward and backward probabilities is that the backward probabilities start the recursive process at the end of the sequence (unlike the forward probabilities which are computed starting at the first observation). Similar to the forward probabilities, the backward probabilities are given by:

$$\beta_n^i = P(O_N, O_{N-1}, \dots, O_n, q_n = i; \lambda) \tag{3.70}$$

The algorithm for computation of the backward probabilities is similar to that for the forward probabilities and is given below:

1. **Initialization:**
$$\beta_T^i = 1 \quad for\ 1 \leq i \leq S \tag{3.71}$$

2. **Induction Step:**
$$\beta_n^i = \sum_{j=1}^{S} (a_{ji}\beta_{n+1}^j b_n^j) \quad for\ n = N-1, N-2, \ldots, 1 \quad for\ 1 \leq i \leq S \tag{3.72}$$

Both the forward and backward probabilities are used extensively in the expectation and maximization steps of the Baum-Welch algorithm presented in the next two sections.

3.4.3 Expectation Step

In the expectation step of the Baum-Welch re-estimation algorithm, several likelihood variables are computed and used for the re-estimation of the left-to-right HMM parameters during the Maximization step. First we introduce a variable representing the probability of the hidden state q_n at time index n to be labeled as i and the hidden state q_{n+1} at time $n+1$ to be labeled as j given the observation sequence and a prior assignment of the HMM parameters λ. This probability is shown below:

$$\xi(i,j,n) = P(q_n = i, q_{n+1} = j | O; \lambda) = \frac{P(q_n = i, q_{n+1} = j, O; \lambda)}{P(O; \lambda)} \tag{3.73}$$

The probability $\xi(i,j,n)$ in (3.73) can be written in terms of the forward probabilities, backward probabilities, hidden state transition probabilities and the emission probabilities as follows:

$$\xi(i,j,n) = \frac{\alpha_n^i a_{ji} b_{n+1}^j \beta_{n+1}^j}{\sum_{i=1}^{S} \sum_{j=1}^{S} \alpha_n^i a_{ji} b_{n+1}^j \beta_{n+1}^j} \tag{3.74}$$

where the numerator in (3.74) is simply the probability $P(q_n = i, q_{n+1} = j, O; \lambda)$ and the denominator in (3.74) is the likelihood of the observation set $P(O; \lambda)$. Let $\gamma(i,n)$ represent the probability of the hidden state q_n at time index n being labeled as i and we can then write it in terms of the forward and backward probabilities as shown below

$$\gamma(i,n) = \frac{\alpha_n^i \beta_{n+1}^i}{\sum_{i=1}^{S} \alpha_n^i \beta_{n+1}^i} = \frac{P(q_n = i, O; \lambda)}{P(O; \lambda)} \tag{3.75}$$

The probability $\gamma(i,n)$ can also be obtained by simply marginalizing the probability $\xi(i,j,n)$ over the state j as shown below:

$$\gamma(i,n) = \sum_{j=1}^{S} \xi(i,j,n) \tag{3.76}$$

Finally, we define the variable $v(i,n,l)$, which represents the probability of the state q_n at the time index n having a label i and being generated by the mixture l from the set of K mixtures in the GMM model for the hidden state, given the observation set and the HMM model λ. This probability is given by:

$$v(i,n,l) = \frac{\alpha_n^i \beta_{n+1}^i}{\sum_{i=1}^{S} \alpha_n^i \beta_{n+1}^i} \frac{c_{il} N(\mu_{il}, \Sigma_{il}, O_n)}{\sum_{k=1}^{K} c_{ik} N(\mu_{ik}, \Sigma_{ik}, O_n)} \tag{3.77}$$

where c_{il}, μ_{il}, and Σ_{il} represent the weight, mean vector, and the covariance matrix of the state i and mixture l. Note that the first product term in (3.77) is recognized from (3.75) to simply be the probability $\gamma(i,n)$. The maximization step of the Baum-Welch algorithm iteration uses the expected values of the aforementioned likelihoods for the re-estimation of the parameters of the left-to-right HMM. The expected number of transitions from the hidden state q_i to the hidden state q_j is given by:

$$E[\xi] = \sum_{n=1}^{N} \xi(i,j,n) \tag{3.78}$$

Similarly the expected number of transitions from state i is given by:

$$E[\gamma] = \sum_{n=1}^{N} \gamma(i,n) \tag{3.79}$$

The expected number of times the observations in the observation sequence will be generated from the mixture l in the hidden state i is given by:

$$E[v] = \sum_{n=1}^{N} v(i,n,l) \tag{3.80}$$

Finally, the variable

$$E_i[v] = \sum_{k=1}^{K} \sum_{n=1}^{N} v(i,n,k) \tag{3.81}$$

represents the expected number of times the observation will be generated from the hidden state i. We now have all the variables needed for the computation of the HMM parameters in the maximization step of a single Baum-Welch iteration.

3.4.4 Maximization Step

We now present the set of computations that is executed to update the left-to-right HMM parameters, namely the hidden state transition probability matrix A and the GMM parameters: weights, mean vectors, and covariance matrixes c_{il}, μ_{il}, and Σ_{il} respectively. Let the observation sequence be denoted by $O = [O_1, O_2, \ldots, O_N]$ and the corresponding hidden state be denoted by $Q = [q_1, q_2, \ldots, q_N]$ for a left-to-right HMM. For each transition from state i to state j the transition probabilities are computed by dividing the expected number of transitions from state i to state j by the expected number of transitions from state i. Thus the updated transition probabilities are given by:

$$a_{ij} = \frac{E[\xi]}{E[\gamma]} = \frac{\sum_{n=1}^{N} \xi(i,j,n)}{\sum_{n=1}^{N} \gamma(i,n)} \tag{3.82}$$

Similarly the weights of the hidden state HMM are given by:

$$c_{il} = \frac{E[v]}{E_i[v]} = \frac{\sum_{n=1}^{N} v(i,n,l)}{\sum_{k=1}^{K} \sum_{n=1}^{N} v(i,n,k)} \tag{3.83}$$

where $\sum_{n=1}^{N} v(i,n,l)$ is the expected number of times the observations will come from mixture component l in the hidden state i and $\sum_{k=1}^{K} \sum_{n=1}^{N} v(i,n,k)$ is the expected number of times the observations will be generated from hidden state i. The updated mean of the distribution is given by the weighted time average of the observation data set, weighted by the likelihood of the observations in the hidden state i being generated from the mixture component l. Thus the mean vector of the GMM mixture l in the hidden state i is given by:

$$\mu_{il} = \frac{\sum_{t=1}^{N} (E[v]) o_t}{E[v]} = \frac{\sum_{t=1}^{N} \left(\sum_{n=1}^{N} v(i,n,l) \right) o_t}{\sum_{n=1}^{N} v(i,n,l)} \tag{3.84}$$

Finally, the covariance matrix is computed in the same manner (as the mean μ_{il}) and is given by:

$$\Sigma_{il} = \frac{\sum_{t=1}^{N}(E[v])(o_t - \mu_{il})(o_t - \mu_{il})^T}{E[v]} \tag{3.85}$$

$$= \frac{\sum_{t=1}^{N}\left(\sum_{n=1}^{N} v(i,n,l)\right)(o_t - \mu_{il})(o_t - \mu_{il})^T}{\sum_{n=1}^{N} v(i,n,l)} \tag{3.86}$$

Since the Baum-Welch re-estimation algorithm is applied to a left-to-right HMM, the initial weights π of the hidden states are one for the first state, and zero for the rest of the states. Note that so far we have presented the equations for updating the parameters of the left-to-right HMM using only one observation sequence. It has been shown that the parameters of the HMM need to be adjusted based on multiple observation sets for the phoneme recognition system, in order to diminish some of the effects of speech and speaker variability. Since the numerators and the denominators of the update equations are averages, we can incorporate multiple observation sequences by simply summing the numerators and the denominators of the update equation for a_{ij}, c_{il}, μ_{il}, and Σ_{il} in (3.82) to (3.86) over all of the observation sequences. Let the entire set of P observation sequences be denoted by $O = [O_1^{l_1}, O_2^{l_2}, \ldots, O_P^{l_P}]$ having a total length N of $N = \sum_{i=1}^{P} l_i$ where $O_i^{l_i}$ denotes the observation sequence i heaving length l_i. Let the likelihood variables $\xi(i,j,n)$, $\gamma(i,n)$, and $v(i,n,l)$ computed in the expectation step of the Baum-Welch algorithm iteration be denoted by $\xi^m(i,j,n)$, $\gamma^m(i,n)$, and $v^m(i,n,l)$ respectively for the observation sequence m. The update equations for the Maximization step of the Baum-Welch iteration are given by:

$$a_{ij} = \frac{\sum_{m=1}^{P}\sum_{n=1}^{N}\xi^m(i,j,n)}{\sum_{m=1}^{P}\sum_{n=1}^{N}\gamma^m(i,n)} \tag{3.87}$$

$$c_{il} = \frac{\sum_{m=1}^{P}\sum_{n=1}^{N} v^m(i,n,l)}{\sum_{m=1}^{P}\sum_{k=1}^{K}\sum_{n=1}^{N} v^m(i,n,k)} \tag{3.88}$$

$$\mu_{il} = \frac{\sum_{m=1}^{P}\sum_{t=1}^{M}\left(\sum_{n=1}^{N} v^m(i,n,l)\right) O_m^{l_m}(t)}{\sum_{m=1}^{P}\sum_{n=1}^{N} v^m(i,n,l)} \tag{3.89}$$

$$\Sigma_{il} = \frac{\sum_{m=1}^{P}\sum_{t=1}^{M}\left(\sum_{n=1}^{N} v^m(i,n,l)\right)(O_m^{l_m}(t) - \mu_{il})(O_m^{l_m}(t) - \mu_{il})^T}{\sum_{m=1}^{P}\sum_{n=1}^{M} v^m(i,n,l)} \tag{3.90}$$

3.5 Practical Issues with HMM

There are a few considerations that need to be addressed, particularly for the re-estimation of the left-to-right HMM parameters, in order for the implementation of the training and recognition of the HMM models of the phoneme recognition system to be feasible. From (3.68) and (3.72) of the recursive algorithms for the computations of the forward and backward probabilities we observe that the values of the forward and backward probabilities converge exponentially to zero. In other words, for observation sequence length N the forward and backward probabilities will be very close to zero. This, in turn, will introduce large rounding errors during the implementation of the training system for the HMM parameters. The rounding errors will be generated from the very small probabilities being rounded to zero. This in turn will leave most of the forward and backward probabilities as zeroes, which will severely degrade the performance of the Baum-Welch algorithm. To eliminate these rounding errors, scaling of the forward and backward probabilities is necessary.

During the computation of the forward and backward probabilities, their values are scaled by scaling coefficients that are not dependent on the hidden states i and are only dependent on the time index n. The goal of these coefficients is to maintain the forward and backward probabilities within the dynamic range of the processor used for the implementation of the Baum-Welch re-estimation algorithm. After the computation of the HMM parameters, the scaling coefficients of the forward and backward probabilities cancel each other. Thus the HMM parameter estimation is not affected by the scaling coefficients. To illustrate this point consider the re-estimation of the transition probabilities a_{ij} in terms of the forward and backward probabilities:

$$a_{ji} = \frac{\sum_{n=1}^{N} \alpha_n^i a_{ji} \beta_{n+1}^j b_{n+1}^j}{\sum_{j=1}^{M} \sum_{n=1}^{N} \alpha_n^i a_{ji} \beta_{n+1}^j b_{n+1}^j} \tag{3.91}$$

The scaling coefficients for the forward probabilities are chosen to be:

$$c_n = \frac{1}{\sum_{i=1}^{M} \alpha_n^i} \tag{3.92}$$

which effectively ensures that the sum of the forward probabilities for time index n is unity.

The scaling of the forward probabilities is performed as follows. First, each of the forward probabilities α_n^i for time index n is computed by the recursive formulation given in (3.68) by using the scaled forward probability for time index $n-1$ as shown below:

$$\alpha_n^i = \sum_{j=1}^{M} \widehat{\alpha_{n-1}^j} a_{ij} b_n^i \tag{3.93}$$

where $\widehat{a_n^j}$ represents the scaled forward probability for state j at time index n. Second, the forward probability computed in (3.93) is scaled by the scaling coefficient defined in (3.92) and is given by:

$$\widehat{a_n^i} = \frac{\sum_{j=1}^{M} \widehat{\alpha_{n-1}^j} a_{ij} b_n^i}{\sum_{i=1}^{M} \sum_{j=1}^{M} \widehat{\alpha_{n-1}^j} a_{ij} b_n^i} \tag{3.94}$$

By induction we have:

$$\widehat{a_{n-1}^i} = (\prod_{k=1}^{n-1} c_k) \alpha_{n-1}^i \tag{3.95}$$

and, by substituting (3.95) into (3.94) we have:

$$\begin{aligned}\widehat{\alpha_n^i} &= \frac{\sum_{j=1}^{M} (\prod_{k=1}^{n-1} c_k) \alpha_{n-1}^i a_{ij} b_n^i}{\sum_{i=1}^{M} \sum_{j=1}^{M} (\prod_{k=1}^{n-1} c_k) \alpha_{n-1}^i a_{ij} b_n^i} &(3.96)\\ &= \frac{\alpha_n^i}{\sum_{i=1}^{M} \alpha_n^i} &(3.97)\end{aligned}$$

Equation (3.97) shows that scaling the forward coefficients in the recursive manner shown in (3.93) (and as implemented for the Baum-Welch re-estimation algorithm) is theoretically equivalent to computing the unscaled forward probabilities a_n^i and then scaling them by the scaling coefficients c_n. The backward probabilities are scaled in the same manner as the forward probabilities, however the scaling coefficients used for the scaling of the backward probabilities at time index n are the same scaling coefficients used for scaling the forward probabilities. Thus,

$$\widehat{\beta_n^i} = c_n \beta_n^i \tag{3.98}$$

Since the magnitudes of the forward and backward probabilities are comparable, scaling the backward probabilities with the same scaling coefficients used for scaling the forward probabilities will be sufficient for maintaining the range of values of the backward probabilities within practical bounds. Let C_n represent the scaling coefficient of the forward probability α_n^i at time index n and D_n represent the scaling coefficient of the backward probability β_n^i at time index n. The scaled forward and backward probabilities are given by:

$$\widehat{\alpha_n^i} = (\prod_{k=1}^{n} c_k)\alpha_n^i = C_n\alpha_n^i \tag{3.99}$$

$$\widehat{\beta_{n+1}^i} = (\prod_{k=n+1}^{N} c_k)\beta_{n+1}^i = D_{n+1}\alpha_n^i \tag{3.100}$$

The re-estimation equation for the transition probabilities using the scaled forward and backward probabilities is given by:

$$a_{ij} = \frac{\sum_{n=1}^{N} C_n\alpha_n^i a_{ij} D_{n+1} b_{n+1}^j b_{n+1}^j}{\sum_{j=1}^{M}\sum_{n=1}^{N} C_n\alpha_n^i a_{ij} D_{n+1} b_{n+1}^j b_{n+1}^j} \tag{3.101}$$

where the product $C_n D_{n+1}$ both in the numerator and denominator of (3.101) is given by:

$$C_n D_{n+1} = \prod_{k=1}^{n} c_k \prod_{k=n+1}^{N} c_k = C_T \tag{3.102}$$

and does not depend on the time index n. As such, the scaling coefficients cancel and do not have any contribution to the re-estimated transition probabilities. Similarly, the scaling of the forward and backward coefficients does not affect the final result of the computation of the variables $\xi(i,j,n)$, $\gamma(i,n)$, and $v(i,n,l)$ in the Expectation step and the HMM parameters c_{il}, μ_{il}, and Σ_{il} in the Maximization step of the Baum-Welch re-estimation algorithm. The only change that is reflected in the Baum-Welch algorithm from scaling the forward and backward probabilities is the computation of the likelihood of the observation sequence $P(O;\lambda)$. After scaling the forward and backward probabilities, we can not simply sum the forward probabilities at time index N to obtain the likelihood of the observation sequence as we did in (3.69). However from (3.100) we can deduce the following:

$$(\prod_{n=1}^{N} c_n)\sum_{i=1}^{S} \alpha_N^i = C_T P(O;\lambda) = 1 \tag{3.103}$$

Thus,

$$P(O;\lambda) = \frac{1}{\prod_{n=1}^{N} c_n} \tag{3.104}$$

The likelihood of the observation sequence as given in (3.104) is prone to numerical overflows and instabilities for large N. Thus we compute the log likelihood of the observation sequence given by:

$$\log(P(O;\lambda)) = -\sum_{n=1}^{N} \log(c_n) \tag{3.105}$$

Note that the forward and backward probabilities do not have to be scaled at each n and can be scaled only for certain time indices as found necessary (for preventing numerical rounding and overflow errors). For the time indices for which the scaling of the forward and backward probabilities is not used, the scaling coefficient is simply 1 and the output is not affected by the scaling coefficient. Furthermore the scaling coefficients can be chosen arbitrarily (while preventing numerical instabilities in the forward and backward probabilities), and thus do not have to be limited by the expression for c_n in (3.92). The only limiting constraint for the choice of the scaling coefficients is that there can be only one scaling coefficient at each time index n.

During the Baum-Welch re-estimation procedure, numerical instabilities can also be generated from having GMM and an observation sequence with high dimensionality D. Assuming independence between each of the dimensions of the mixtures in the GMM, the variable $v(i,n,l)$ is given by:

$$v(i,n,l) = \frac{\alpha_n^i \beta_{n+1}^i}{\sum_{i=1}^{M} \alpha_n^i \beta_{n+1}^i} \frac{c_{il} \prod_{d=1}^{D} N(O_n;\mu_{ik},\Sigma_{ik})}{\sum_{k=1}^{K} c_{ik} \prod_{d=1}^{D} N(O_n;\mu_{ik},\Sigma_{ik})} \tag{3.106}$$

where D is the dimensionality of the feature vectors, and μ_{ild}, σ_{ild}, and $O_{n,d}$ are the mean, variance, and observation respectively for the state i mixture l and along the dimension d. To eliminate the numerical instabilities from the high dimensionality D of the observation data we scale each of the probabilities $N(O_n;\mu_{ik},\Sigma_{ik})$ by a scaling factor s_n^j that is dependent on the time index n and the state j and is independent of the dimension index d and the mixture component l. These constraints ensure that the scaling of the Gaussian probabilities will not affect the computation of the variable $v(i,n,l)$. The scaling coefficient s_n^j is given by:

$$s_n = \frac{1}{med(N(O_n;\mu_{ik},\Sigma_{ik}))} \quad for\ 1 \leq d \leq D,\ 1 \leq l \leq K,\ 1 \leq i \leq S \tag{3.107}$$

where the function $med(\cdot)$ represents the median of a set.

The algorithms integrated in the phoneme recognition system were discussed in this chapter. The next chapter presents the simulation analysis and validation of the functionality of the algorithms discussed in this chapter. Furthermore, in the next chapter we discuss the application of the algorithms discussed in this chapter to the task of phoneme recognition.

Chapter 4

Simulations, Tests, and Results

This chapter presents simulation and testing results of the algorithms discussed in the previous chapter. The set of simulations and test experiments illustrated in this chapter validate the correct functionality and asses the performance of the two recognition layers (first and second layer with left-to-right HMM and segmental ergodic HMM respectively) individually. Particularly we present the performance of the K-Means clustering, EM Clustering, Baum-Welch and Viterbi Search algorithms of the simple left-to-right HMM. We also present the training and the Segmental Viterbi Search algorithm of the ergodic Segmental HMM and finally the performance of the phoneme recognition system.

This chapter also presents one more algorithm that is used for validating the effectiveness of the extracted feature vectors for each phoneme. The validation of the feature vector extraction system assumes that the set of feature vectors generated from two different phonemes with different acoustic properties will present two different clusters when visualized together. For visualizing two sets of feature vectors (from two different phonemes) we need to scale the dimensionality of the concatenated sets of feature vectors to two dimensions, since each of the feature vectors extracted from the speech utterance is a 25 dimensional vector. The dimensionality reduction of the data is performed by the Multidimensional Scaling (MDS) algorithm. The goal of the MDS algorithm is to preserve the variance of the higher dimensional feature vector data in the lower dimensional feature vector data.

Section 1 of this chapter discusses and tests the MDS algorithm, Section 2 presents the simulations to verify the performance of the K-Means and EM clustering algorithms, Section 3 and 4 present the simulations of the left-to-right HMM and the ergodic Segmental HMM respectively. Finally, Section 5 presents the validation of the feature vector system as well as the experimental design and the verification of the phoneme recognition system as a whole.

4.1 Multidimensional Scaling

Multidimensional Scaling refers to a set of techniques that is generally used for visualization of the underlying structure of high dimensional data. The dimensionality reduction (or MDS) techniques used on a high-dimensional data set can also effectively increase the performance of a classifier by avoiding the curse of dimensionality. We use MDS for testing and verification of the algorithms discussed in the previous chapter and of the feature extraction system. The MDS verifies the correct functionality of an algorithm with high dimensional output data by scaling the dimensionality of the output data. After the output of a system is scaled to two dimensions, we are able to visualize the output of the system and compare it to the expected output structure. The application of the MDS techniques to the validation of the algorithms described in the previous chapters is included in the following sections of this chapter. We now proceed with a discussion of the dimensionality reduction itself.

Depending on the application of the MDS and the nature of the original data, different measures of similarity (or proximity between the higher and lower dimensional data) can be applied. The dimensionality reduction used in this thesis is referred to as *classical scaling* because of the fact that the similarity measure used in the MDS algorithm is exactly the Euclidian Distance [25]. The algorithm used for the dimensionality reduction of high dimensional data is the SMACOF (Scaling by MAjorizing a COmplicated Function). As the name implies, the SMACOF algorithm is essentially based on the general concept of majorization of a function. Before we proceed to the details of implementation of the SMACOF, we first introduce the principle behind the majorization of a function.

The principle of majorization is to construct an additional function that majorizes an objective function. Suppose we have an objective function $f(x)$ that needs to be minimized. The analytical solution of the minimization of $f(x)$ could be complicated and cumbersome. Instead the majorization of $f(x)$ recommends a surrogate function $g(x, y)$ for all x:

$$g(x, y) \geq f(x) \tag{4.1}$$

where y is a fixed constant and is called the *supporting point* [25]. The surrogate function $g(x, y)$ is equal to the objective function $f(x)$ when $x = y$. Thus,

$$g(y, y) = f(y) \tag{4.2}$$

For a value of $x_{min} = \operatorname{argmin}_x g(x, y)$ the following inequality holds (known as the *sandwich inequality*):

$$f(x_{min}) \leq g(x_{min}, y) \leq g(y, y) = f(y) \tag{4.3}$$

Majorization of the objective function is an iterative procedure and is carried out by the following steps:

1. **Initialize the fixed point** $y = y_o$
2. **Find** $x^{(m)}$ **such that** $g(x^{(m)}, y) \leq g(y, y)$
3. **If** $|f(y) - f(x^{(m)})| < \epsilon$ **then stop. Otherwise, set** $y = x^{(m)}$ **and return to step 2.**

This process is extended to the case of multidimensional reduction, as the sandwich inequality in (4.3) holds. The objective function used in MDS for visualization and validation in this thesis is the sum of squared distances, otherwise known as *stress* [25].

Let the input to the SMACOF algorithm be the high dimensional data represented by an $N \times D_H$ matrix X_{in}, where N is the total number of data points and D_H is the dimensionality of the input data. Similarly let the output data of the SMACOF algorithm be represented by a $N \times D_L$ matrix Y where D_L is the dimensionality of the lower dimensional data. The SMACOF algorithm first constructs a square, $N \times N$ distance matrix Δ of all the distances

ψ_{ij} between the corresponding data points $X_{in}^{(i)}$ and $X_{in}^{(j)}$ (which is also hollow ($\psi_{ii} = 0$) and symmetrical). The goal of the SMACOF algorithm is to generate a set of lower dimensional points X, whose $N \times N$ distance matrix Γ with its distance elements d_{ij} approximates the distance matrix Δ. Thus, after the iterations of the SMACOF algorithm $\Delta \approx \Gamma$, and the corresponding distance matrix elements ψ_{ij} and d_{ij} are given by:

$$\psi_{ij} = \sqrt{\sum_{d=1}^{D_H} (X_{in}^{(i)}(d) - X_{in}^{(j)}(d))^2} \tag{4.4}$$

$$d_{ij} = \sqrt{\sum_{d=1}^{D_L} (X_i(d) - X_j(d))^2} \tag{4.5}$$

The stress function to be minimized for the dimensionality reduction is formulated as follows:

$$\nu(X) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} (\psi_{ij} - d_{ij})^2 \tag{4.6}$$

where, w_{ij} are the elements of an $N \times N$ weight matrix which is assumed to be symmetric, non-negative and hollow ($w_{ii} = 0$). Without loss of generality we assume:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} \psi_{ij} = 1 \tag{4.7}$$

The expanded form of (4.6) is given by:

$$\nu(X) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} \psi_{ij}^2 + \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} d_{ij}^2 - 2 \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} \psi_{ij} d_{ij} \tag{4.8}$$

$$\nu \overset{\text{def}}{=} \eta_\psi^2 + \eta_d^2(X) - 2\rho(X) \tag{4.9}$$

where, η_ψ^2 is the first term in (4.9) and is assumed to be 1 by (4.7), $\eta_d^2(X)$ is the second term in (4.9) and is a quadratic convex function of X. Finally, $\rho(X)$ corresponds to the third term in (4.9) and is also a convex quadratic function of X with the resulting $-2\rho(X)$ being a concave quadratic function. Let us define a square $N \times N$ matrix V with its corresponding elements v_{ij} as,

$$v_{ij} = \begin{cases} -w_{ij} & \forall\, i \neq j \\ \sum_{j=1, i \neq j}^{N} w_{ij} & \forall\, i = j \end{cases} \tag{4.10}$$

Then the quantity $\eta_d^2(X)$ is reduced to $\eta_d^2(X) = tr(X^{'}VX)$. Furthermore, let us define a square $N \times N$ matrix $B(X)$ with its corresponding elements b_{ij} as,

$$b_{ij} = \begin{cases} -\frac{w_{ij}\psi_{ij}}{d_{ij}(X)} & \forall\, i \neq j,\ d_{ij}(X) > 0 \\ 0 & \forall\, i \neq j\ d_{ij}(X) = 0 \\ \sum_{j=1, i\neq j}^{N} b_{ij} & \forall\, i = j \end{cases} \tag{4.11}$$

With this definition of the matrix $B(X)$, the quantity $\rho(X)$ reduces to $\rho(X) = tr(X^{'}B(X)X)$ and consequently the stress function $\nu(X)$ becomes:

$$\nu(X) = 1 + tr(X^{'}VX) - 2tr(X^{'}B(X)X) \tag{4.12}$$

Similarly, let us define the matrix $B(Y)$ corresponding to the $N \times D_L$ matrix Y consisting of a set of D_L dimensional vectors, whose elements b_{ij} are given by:

$$b_{ij} = \begin{cases} -\frac{w_{ij}\psi_{ij}}{d_{ij}(Y)} & \forall\, i \neq j,\ d_{ij}(Y) > 0 \\ 0 & \forall\, i \neq j,\ d_{ij}(Y) = 0 \\ \sum_{j=1, i\neq j}^{N} b_{ij} & \forall\, i = j \end{cases} \tag{4.13}$$

By the Cauchy-Schwartz inequality we can deduce that $tr(X^{'}B(X)X) \geq tr(X^{'}B(Y)Y)$ [25]. We can consider Y to represent the fixed point in (4.2) (in this case a point y in (4.2) is represented by a set of vectors), as such, the SMACOF algorithm majorizes the quadratic function $\rho(X) = tr(X^{'}B(X)X)$ by a linear function $tr(X^{'}B(Y)Y)$. The stress measure is also majorized in the same manner as shown below.

$$\nu(X) = 1 + tr(X^{'}VX) - 2tr(X^{'}B(X)X) \tag{4.14}$$

$$\leq 1 + tr(X^{'}VX) - 2tr(X^{'}B(Y)Y) = \tau(X, Y) \tag{4.15}$$

where, $\tau(X, Y)$ is a (simplified) quadratic function of X that majorizes the stress measure $\nu(X)$. To find the minimum of the majorization function $\tau(X, Y)$ we need to analytically solve for X in the first partial derivative of $\tau(X, Y)$ with respect to X. This is illustrated below:

$$X_{min} = X \quad s.t. \quad \frac{\partial \tau(X, Y)}{\partial X} = 2VX - 2B(Y)Y = 0 \tag{4.16}$$

To solve (4.16) for X_{min}, we first take the Moore-Penrose inverse V^+ of V. X_{min} is then given by:

$$X_{min} = V^+ B(Y)Y \tag{4.17}$$

The SMACOF algorithm implementation in this thesis uses the weights $w_{ij} = 1$ for all $i \neq j$. Thus the solution for X_{min} reduces to $X_{min} \approx N^{-1}B(Y)Y$. As mentioned before, the majorization is an iterative procedure, whose steps iteratively find the optimal solution X_{min}. The monotone convergence of this iterative majorization procedure is guaranteed by at least a linear convergence rate [25]. The iterative process of majorization formulated for the task of dimensionality reduction is formally stated below:

1. **Initialize the set of lower dimensional vectors given by the $N \times D_L$ matrix $Y = X_o$ and set $m = 0$:**

   The initial set of lower dimensional vectors Y - in this thesis - is simply generated by taking the first D_L dimensions of the higher dimensional data X_{in}.

2. **Find $X_{min}^{(m+1)}$ by the equation $X_{min}^{(m+1)} = N^{-1}B(Y^{(m)})Y^{(m)}$**

3. **If $|\nu(X_{min}^{(m+1)}) - \nu(X_{min}^{(m)})| < \epsilon$ or a certain number of iterations has been reached then stop. Otherwise, set $Y = X_{min}^{(m+1)}$, $m = m + 1$ and return to step 2.**

We have discussed the application of the majorization to the MDS and the corresponding SMACOF algorithm. Now we present the simulation and the verification of the functionality of the MDS algorithm. The MDS is simulated by providing two 25 dimensional input data to the SMACOF algorithm and plotting the corresponding 2 dimensional output data. Both input data sets correspond to random data that is generated from multiple Gaussian distributions with a set of predetermined means and covariance matrixes. The first input 25 dimensional data is generated randomly from 10 different Gaussian distributions. Means and covariances of those Gaussian distributions are predetermined by randomly choosing (from a uniform distribution) a set of mean vectors and covariance matrices from a given range of values. Thus there is a maximum threshold for the values in the mean vectors and a separate maximum threshold chosen for the values in the covariance matrixes. Those thresholds are 100 and 1750 respectively. The second input data was generated from 5 different Gaussian distributions. Figures 4.1 and 4.2 show the 2 dimensional outputs of the SMACOF algorithm (for the two 25 dimensional data sets) as well as the change in the stress in each iteration. Tables 4.1 and 4.2 list all the test parameters that are used in the first and second simulations of the MDS algorithm respectively.

Table 4.1: Parameters for the first MDS experiment (results shown in Figure 4.1)

Parameter	**Value**	**Description**
M	10	Number of clusters in the test data.
N	4000	Number of test data points.
D_{in}	25	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the MDS algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take
Covariance Threshold	1750	Maximum possible value that an element in a covariance matrix can take

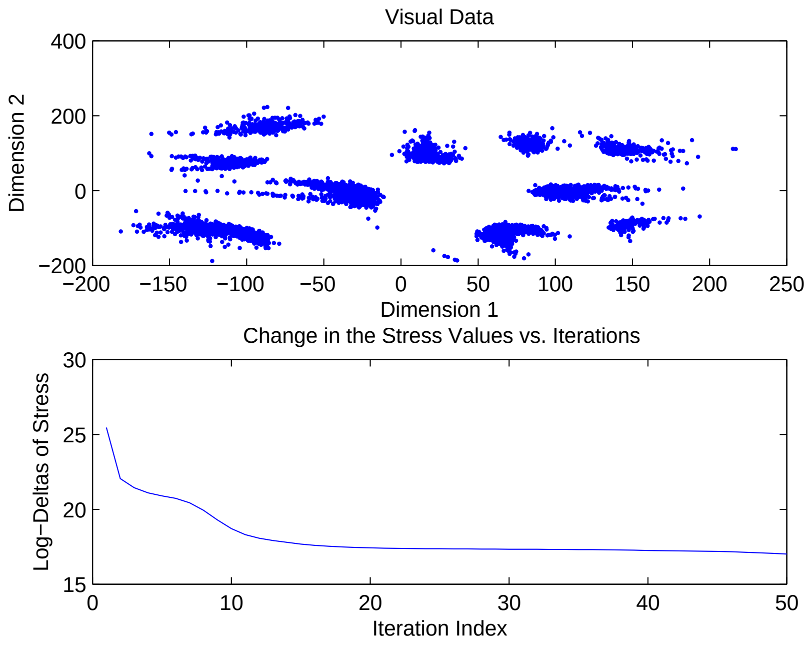

Figure 4.1: MDS Scaled output and convergence of 25 dimensional input data with 10 clusters.

Table 4.2: Parameters for the second MDS experiment (results shown in Figure 4.2)

Parameter	**Value**	**Description**
M	5	Number of clusters in the test data.
N	4000	Number of test data points.
D_{in}	25	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the MDS algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take
Covariance Threshold	1750	Maximum possible value that an element in a covariance matrix can take

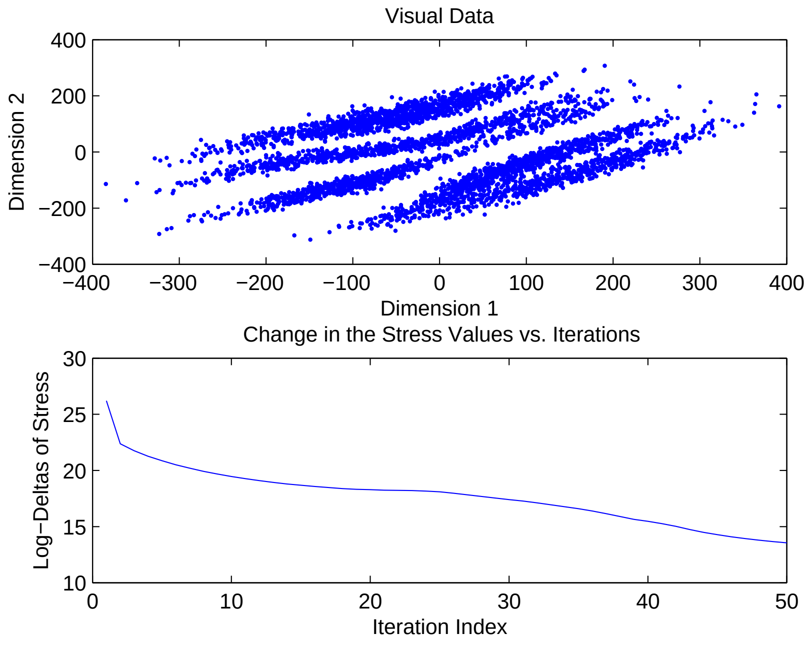

Figure 4.2: Scaled output and convergence of 25 dimensional input data from 5 clusters.

The top plots in Figures 4.1 and 4.2 correspond to the 2 dimensional output of the SMACOF algorithm at the end of 50 iterations. On the other hand, the bottom plots of Figures 4.1 and 4.2 show the change in the stress values $\nu(X^{(m)})$ for the iteration index m. As we see from the top plots of Figures 4.1 and 4.2, the number of clusters of the scaled 2 dimensional data is 10 and 5 respectively. This experiment shows that the underlying structure of the high dimensional data is indeed preserved in the lower dimensional space.

Furthermore, Figures 4.1 and 4.2 show that the two dimensional output data does not have an elliptical pattern (as does 2 dimensional data generated according to a Normal distribution) and is morphed. This is probably due to the fact that the output data has been projected from a 25 dimensional space to a 2 dimensional space. As we see from the bottom plots of Figures 4.1 and 4.2 the change in the stress values $\nu(X^{(m)})$ of the SMACOF algorithm indeed decreases monotonically, as expected. Thus Figures 4.1 and 4.2 illustrate correct functionality of the MDS using the SMACOF algorithm. The next section demonstrates the simulation and test results of the clustering algorithms described in the previous chapter.

4.2 Simulation and Tests of Data Clustering

This section of the thesis presents the simulation, testing, and verification of the two clustering algorithms discussed in the previous chapter, namely, K-Means clustering and EM clustering. Numerous simulation data sets were clustered for the verification of the performance of three tasks. First, the initialization of the K-Means algorithm was verified to have the class centers (chosen as data points from the data set) to be as far apart from each other as possible. Second, the functionality of the K-Means algorithm is illustrated by looking at the positions of the final cluster centers after the convergence of the algorithm. Finally, the functionality of the EM clustering algorithm is demonstrated by showing the final cluster center and the shape and the orientation of the probability contours. The probability contours provide one way to visualize the variance and the covariance parameters estimated by the EM Clustering algorithm. The following subsections discuss the detailed experimental setup, testing, and simulation results for the K-Means and EM clustering algorithms separately.

4.2.1 K-Means

The K-Means initialization and the K-Means algorithms are tested simultaneously in this subsection. The input data generation for the K-Means algorithm was done in the same manner as for the MDS tests. The test data was generated randomly from a single or multiple Normal distributions. The mean vectors and the covariance matrices of the Gaussian distributions where randomly chosen from a uniform distribution, given the maximum thresholds (for the values in the mean vectors and the covariance matrixes). The randomly generated covariance matrices where multiplied by their corresponding transposes, resulting in a non-negative definite symmetric matrices. The maximum threshold for the mean vectors and covariance matrices was chosen to be 100 and 20 respectively. The first test data was two dimensional data generated from a single Gaussian. The number of clusters K assigned to the K-Means clustering algorithm was 5. Table 4.3 lists all the parameters used in this experimental setup for testing the K-Means algorithm. Figure 4.3 shows the clustering results, while Figure 4.4 demonstrates the convergence of the centers (estimated by the K-Means algorithm).

Table 4.3: Parameters for the first K-Means experiment (results shown in Figure 4.3)

Parameter	**Value**	**Description**
M	1	Number of clusters in the test data.
K	5	Number of clusters the test data is partitioned into.
N	10000	Number of test data points.
D_{in}	2	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	20	Maximum possible value that an element in a covariance matrix can take.

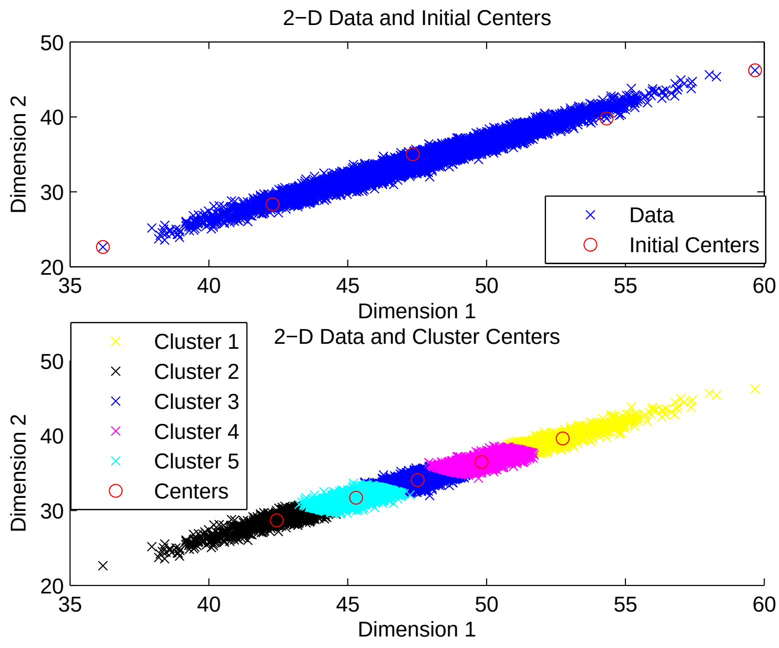

Figure 4.3: K-Means Initialization and K-Means algorithm result for 2 dimensional input data from a single Gaussian and 5 estimated clusters.

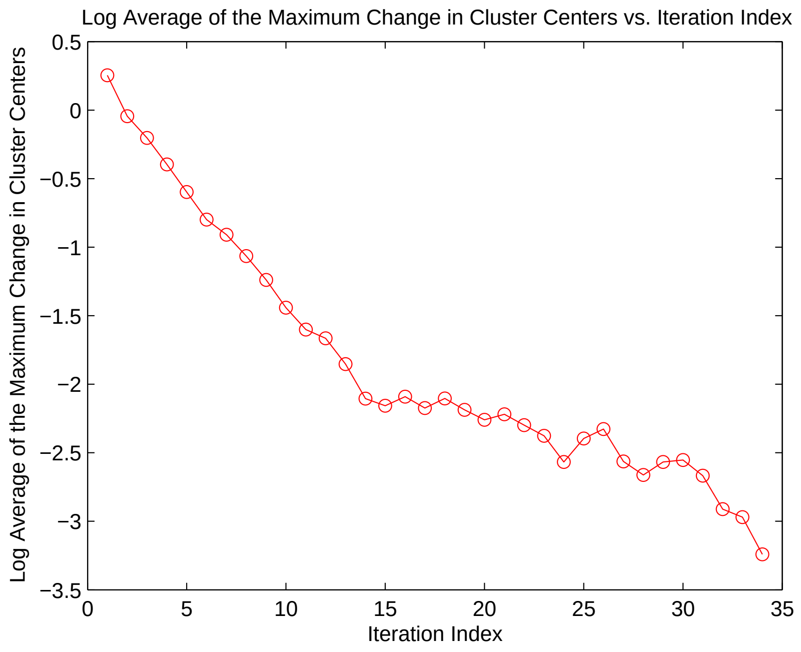

Figure 4.4: Convergence of the K-Means algorithm.

The top plot of Figure 4.3 shows the two dimensional input data and the corresponding initialized centers. We note that the centers are chosen from the data points in the input data. The bottom plot of Figure 4.3 shows input data, the centers after 20 iterations of the K-Means algorithm and the points that are classified to each cluster (based on the center with the shortest Euclidian distance from each point). From the bottom plot in Figure 4.3 we see that the input data has been clustered into five different clusters and that the centers of the clusters are within the input data. Also note from the bottom plot in Figure 4.3 that the cluster at the mean of the input data (Cluster 3) occupies less area than the clusters at the tails of the input data. This is because of the fact that the probability of a data point being near the center of the input data is higher than it being near the tail of the input data. Thus, data points in Cluster 3 are more densely distributed than in Clusters 1 and 2 in Figure 4.3. Figure 4.4 shows the change in the center locations during each iteration. For each iteration, the change in the center locations is computed by first picking the cluster center that had the highest change in value for each dimension. Then the change in the center locations is computed by averaging these values over all the dimensions. As Figure 4.4 illustrates, the K-Means algorithm converges as the number of iterations increases.

The next input data set is a 25 dimensional input generated from a single Gaussian distribution. The remaining parameters are left unchanged from the previous experimental setup. Table 4.4 lists the configuration parameters and their values as used in this experiment. Figure 4.5 shows the initialization of the K-Means algorithm with the corresponding center locations and the clusters resulting after 40 iterations of the K-Means algorithm.

Table 4.4: Parameters for the second K-Means experiment (results shown in Figure 4.5)

Parameter	**Value**	**Description**
M	1	Number of clusters in the test data.
K	5	Number of clusters the test data is partitioned into.
N	4000	Number of test data points.
D_{in}	25	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	250	Maximum possible value that an element in a covariance matrix can take.

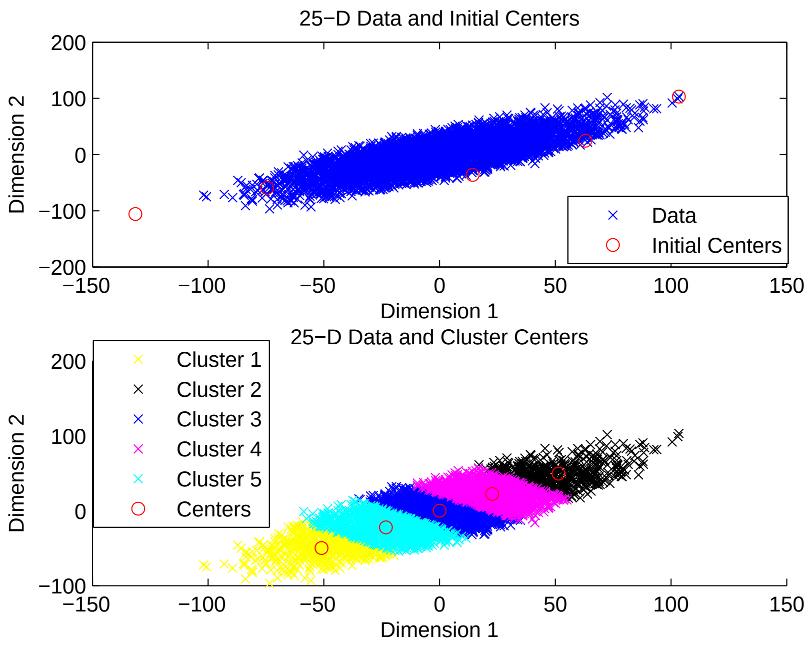

Figure 4.5: K-Means Initialization and K-Means algorithm result for 25 dimensional input data from a single Gaussian distribution and 5 estimated clusters.

Similar to Figure 4.3, the top part of Figure 4.5 shows the scaled data (scaled from 25 dimensions to 2 dimensions using the MDS algorithm described in the previous section) and the results of the K-Means initialization algorithm. The bottom plot in Figure 4.5 illustrates the corresponding scaled results of the K-Means algorithm. Note that the initial centers of the K-Means initialization algorithm seem no longer to be chosen from the input data. This is because of the fact that due to the large number of data points, only a certain number of data points (4000) out of the original input data set (10000 points) were randomly selected with the 25 dimensional initial centers, then scaled (with the SMACOF algorithm) and shown. Thus, the data points corresponding to the initial centers might not be included in the random selection. Also note that the scaled data in Figure 4.5 is not morphed. This is because the 25 dimensional input data was generated according to a single Normal distribution only, and the MDS algorithm (with the Euclidian distance as the similarity measure) preserves the variance of the higher dimensional data set. The bottom of Figure 4.5 illustrates the performance of the K-Means algorithm for a high dimensional input. The observations made on Figure 4.3 are also valid here.

The next experimental setup has 2 dimensional input data generated from 10 different Gaussian distributions. The number of clusters approximated for this input data was also 10. Table 4.5 lists the configuration parameters and values used in this K-Means algorithm simulation. The top and bottom plots in Figure 4.6 show the results of the initialization of the K-Means algorithm and the resulting K-Means centers after 20 iterations respectively.

Table 4.5: Parameters for the third K-Means experiment (results shown in Figure 4.6)

Parameter	**Value**	**Description**
M	10	Number of clusters in the test data.
K	10	Number of clusters the test data is partitioned into.
N	10000	Number of test data points.
D_{in}	2	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	20	Maximum possible value that an element in a covariance matrix can take.

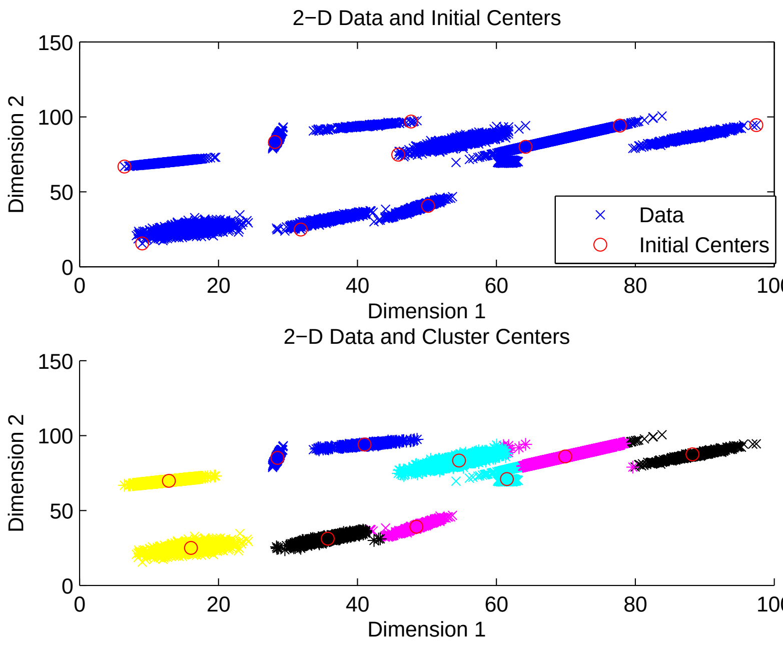

Figure 4.6: K-Means Initialization and K-Means algorithm result for 2 dimensional input data from 10 Gaussian distributions and 10 estimated cluster centers.

As we see from the results of the K-Means initialization algorithm from Figure 4.6 (top plot) the 10 centers were chosen from 10 different data points. Note that these centers are also chosen to be as far from each other as possible, encompassing the entire input data set, as expected. The final centers of the K-Means algorithm (bottom plot of Figure 4.6) match all 10 clusters. This figure shows the effective performance of the K-Means algorithm and its corresponding initialization. However, as we can see from the bottom plot of Figure 4.6, the classification of the data points to their corresponding clusters based on the data center nearest to each point (having smallest Euclidian distance) is not accurate. There are several clusters, close to each other, that have the data points classified incorrectly in Figure 4.6. This is because of the fact that the classification of the data points to their corresponding classes does not take into account the covariance of each cluster. We will demonstrate in the next section that the EM Clustering algorithm significantly increases the correct classification of data points to their clusters due to it also estimating the covariances and weights of the clusters (weights are the number of data points classified to a cluster divided by the total number of data points).

The final experimental setup performed for the K-Means algorithm has 25 dimensional input data generated from a set of 5 Normal distributions. The number of cluster centers estimated by the K-Means algorithm is also 5. Table 4.6 lists the configuration parameters and values used in this K-Means algorithm simulation. The top and bottom plots of Figure 4.7 show the MDS scaled result of the K-Means initialization algorithm and the final scaled cluster centers after 20 iterations of the K-Means algorithm.

Table 4.6: Parameters for the fourth K-Means experiment (results shown in Figure 4.7)

Parameter	**Value**	**Description**
M	5	Number of clusters in the test data.
K	5	Number of clusters the test data is partitioned into.
N	4000	Number of test data points.
D_{in}	25	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	250	Maximum possible value that an element in a covariance matrix can take.

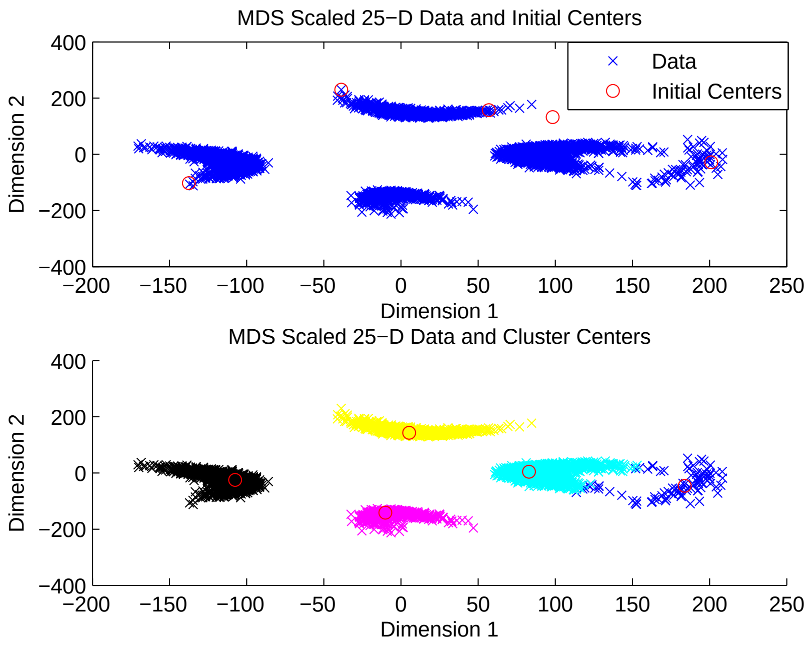

Figure 4.7: K-Means Initialization and K-Means algorithm result for 25 dimensional input data from 5 Gaussian distributions and 5 estimated cluster centers.

We observe in the bottom plot in Figure 4.7 that the clusters of the scaled input data (coming from Normal distributions) are morphed (as expected) and the final class centers are within each of the clusters. Also the input data points (that were selected for scaling) were correctly classified, since the clusters are relatively far apart from each other. Figure 4.7 shows the effective performance of the K-Means initialization and the K-Means algorithm for high dimensional data. In the next subsection of the thesis the experiments and simulations of the EM Clustering algorithm are presented.

4.2.2 EM Clustering

The EM Clustering algorithm was tested in a similar fashion as the K-Means clustering algorithm. The input data was generated according to a single or multiple multivariate Gaussian distribution(s). The EM Clustering algorithm was initialized by the K-Means algorithm as discussed in Section 3.4.1. The experimental results also show the probability contours of the Gaussian probability distributions, whose parameters (being the set of mean vectors and covariance matrices) are estimated by the EM Clustering algorithm. The classification advantages of EM Clustering (or soft clustering) over K-Means clustering (or hard clustering) are also noted in this subsection.

The first experimental setup has 2 dimensional input data randomly generated from 3 different Gaussian distributions. The EM clustering algorithm was performed on this input data set for determining the means, covariances, and the weights of these 3 clusters. The configuration parameters for this experiment are listed in Table 4.7, while the results of the EM clustering algorithm are illustrated in Figure 4.8.

Table 4.7: Parameters for the first EM Clustering algorithm experiment (results shown in Figure 4.8)

Parameter	**Value**	**Description**
M	3	Number of clusters in the test data.
K	3	Number of clusters the test data is partitioned into.
N	10000	Number of test data points.
D_{in}	2	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	20	Maximum possible value that an element in a covariance matrix can take.

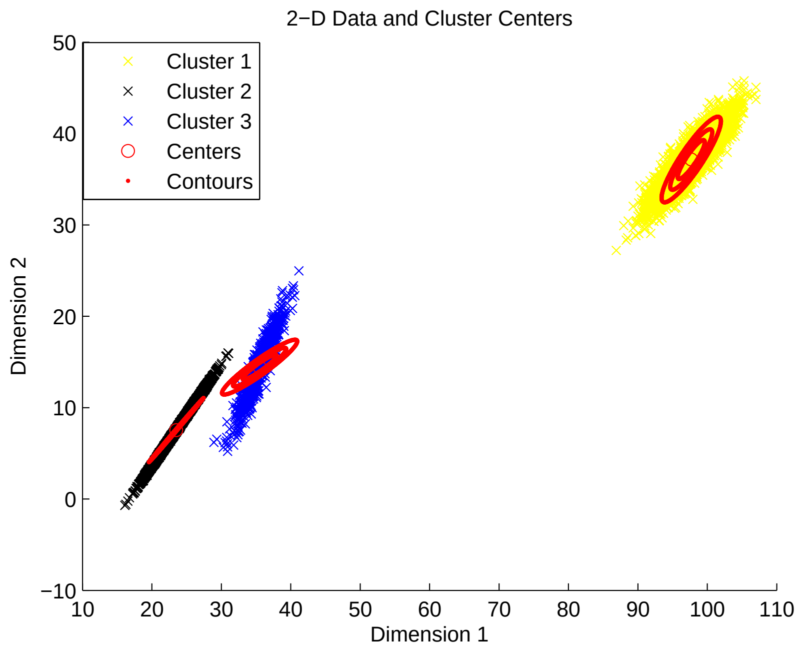

Figure 4.8: EM clustering algorithm results for 2 dimensional input data from 3 Gaussian distributions with estimated parameters for 3 Gaussian mixtures.

As we see from Figure 4.8 the three cluster centers, or cluster means, have been correctly identified. The probability contours of Clusters 1 and 2 in Figure 4.8 follow the shape of the input data. The probability contours of Cluster 3 in Figure 4.8 have the same shape as the input data for that cluster, however the rotation of the probability contours is erroneous, since the covariances between the data dimensions display approximation error for this cluster. Thus, Figure 4.8 illustrates the performance of the EM Clustering algorithm on 3 cluster input data. Figure 4.9 shows the change in the log-likelihood of the observation set for each iteration of the EM clustering algorithm.

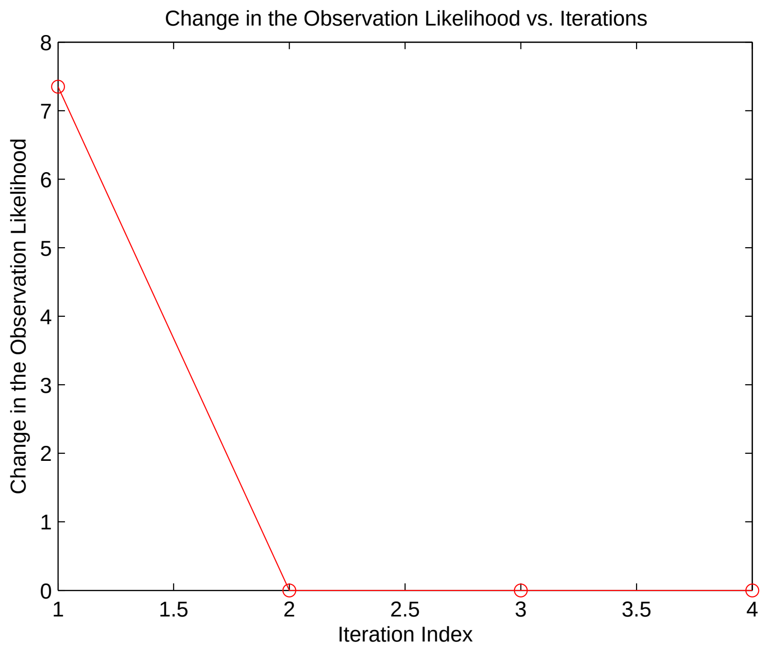

Figure 4.9: Convergence of the EM algorithm for the three-cluster input data.

As observed in Figure 4.9, the likelihood of the observation set indeed converges. The number of iterations sufficient for convergence of the EM algorithm was 2 in this case (the values for iterations 2 through 4 are equal to zero). However, as we see later in this subsection, low within-cluster variance delays the convergence of the EM Clustering algorithm.

The next experimental setup has input data set generated from 10 different Gaussian distributions and the EM Clustering algorithm estimates the corresponding mean vectors, covariance matrices, and weights for each of these 10 clusters (Normally distributed). This simulation was performed in the same manner as in the previous EM Clustering algorithm simulation, and the results were also posted in a similar manner as in Figure 4.8. Table 4.8 lists the parameters used for this test experiment of the EM clustering algorithm, while Figure 4.10 demonstrates the performance of the EM clustering algorithm and shows the corresponding cluster means and probability contours.

Table 4.8: Parameters for the second EM Clustering algorithm experiment (results shown in Figure 4.10)

Parameter	**Value**	**Description**
M	10	Number of clusters in the test data.
K	10	Number of clusters the test data is partitioned into.
N	10000	Number of test data points.
D_{in}	2	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	20	Maximum possible value that an element in a covariance matrix can take.

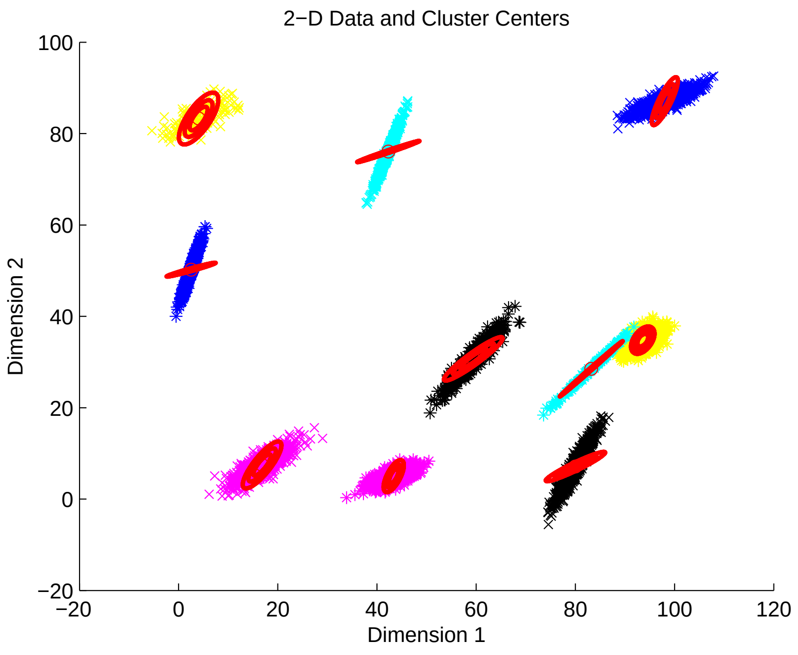

Figure 4.10: EM clustering algorithm results for 2 dimensional input data from 10 Gaussian distributions with estimated parameters for 10 Gaussian mixtures.

As observed in Figure 4.10, all the means of the clusters were estimated correctly. Furthermore, all of the probability contours have the same shape as their corresponding input cluster data, however, some of the Gaussian probability contours have incorrect rotations due to the approximation errors in the Gaussian covariance matrixes. Also, note that even though some of the clusters in Figure 4.10 overlap, the original classification of the points to their corresponding clusters is performed correctly. This is due to the fact that the data is classified to its class according to a probability measure. The classification performance comparison of the EM clustering algorithm and the K-Means clustering algorithm is illustrated in the next set of simulations.

This experiment is conducted to specifically compare performance between the K-Means clustering algorithm and the EM clustering algorithm. The input data used for the simulation of the EM clustering algorithm is the same as the input data used in Figure 4.6. The input data is randomly generated from 10 different Gaussian probabilities as shown in Figure 4.6. The top of Figure 4.11 presents the cluster centers and the input data classification to the classes after the convergence of the K-Means algorithm. On the other hand, the bottom plot of Figure 4.11 shows the means, probability contours, and classification of the input data points to the 10 clusters after the convergence of the EM clustering algorithm. The configuration parameters for this experiment are listed in Table 4.5.

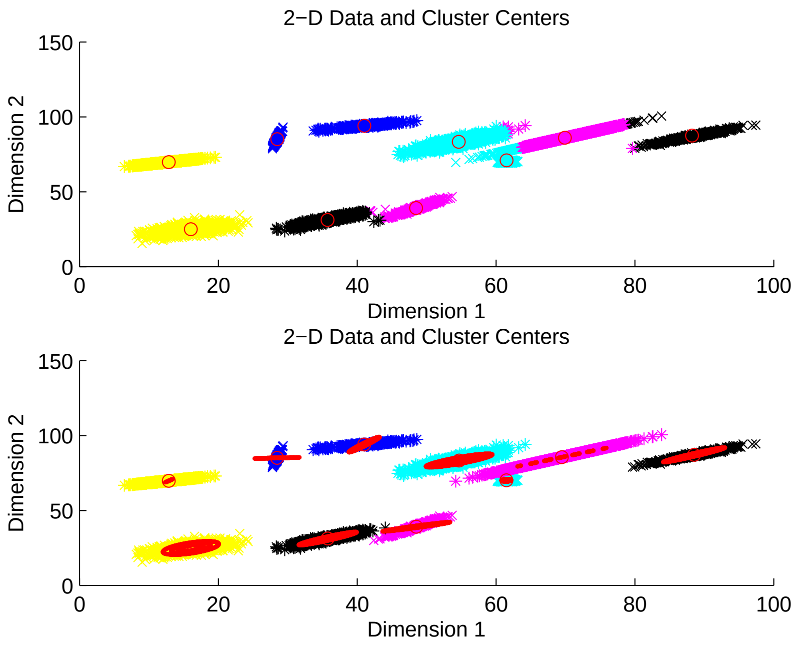

Figure 4.11: Classification performance comparison of the K-Means and the EM algorithms.

As we see from the top plot in Figure 4.11, classification of the data points by the means of the Euclidian distance results in classification errors between clusters that are close to each other. On the other hand, the classification of input data in the bottom plot of Figure 4.11 is performed by assigning the class label with the highest probability to each of the observed data points. The probability measure of the observation, given the cluster label, is based on the mean, covariance, and weight of the class. Hence the classification of the data to the clusters is more accurate after the convergence of the EM clustering algorithm, than after the convergence of the K-Means algorithm, as illustrated in Figure 4.11.

The final experimental setup of the EM clustering algorithm simulation has 2 dimensional input data generated from a single Gaussian distribution. However, there are 5 Gaussian parameter sets estimated by the EM Clustering algorithm. Table 4.9 lists the configuration parameters used for this experiment. Figure 4.12 demonstrates the cluster centers, probability contours, and corresponding distributions of the data points to the clusters, while Figure 4.13 shows the convergence of the change in the observation likelihood of the EM clustering algorithm.

Table 4.9: Parameters for the final EM Clustering algorithm experiment (results shown in Figure 4.12)

Parameter	**Value**	**Description**
M	1	Number of clusters in the test data.
K	5	Number of clusters the test data is partitioned into.
N	10000	Number of test data points.
D_{in}	2	Dimensionality of the input test data.
D_{out}	2	Dimensionality of the output of the scaled K-Means algorithm.
Mean Threshold	100	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	20	Maximum possible value that an element in a covariance matrix can take.

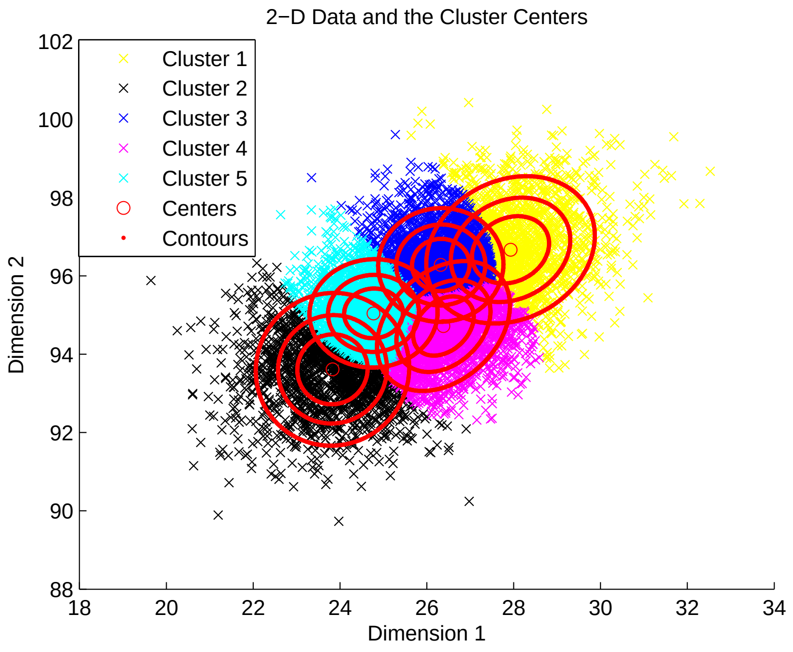

Figure 4.12: EM clustering algorithm results for 2 dimensional input data from a single Gaussian distribution with estimated parameters for 5 Gaussian mixtures.

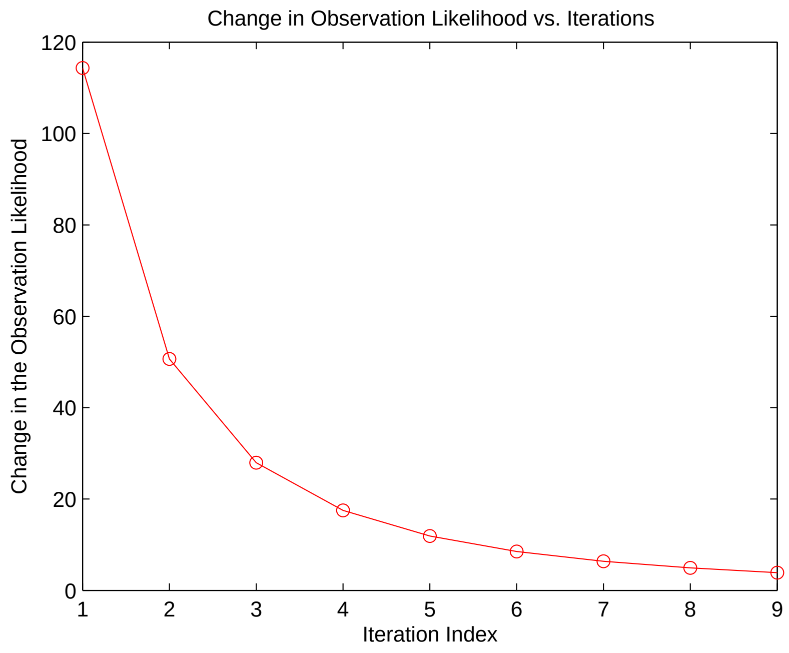

Figure 4.13: Convergence of the EM algorithm for the five-cluster input data.

As we see from Figure 4.12, the estimated clusters, after the convergence of the EM Clustering algorithm, are relatively close to each other. Thus, since the ratio of the cluster covariance and the within-cluster variance is large, we expect more iterations to be required for the convergence of the EM clustering algorithm than when the clusters are far apart from each other (as is the case in Figure 4.8). Figure 4.13 shows that the convergence of the EM Clustering algorithm is in fact slower (about 9 iterations) for given input data with small within-cluster variance than the convergence of the EM algorithm given input data with distinct clusters (as shown in Figure 4.9). The next section of the thesis presents the simulation and test analysis of the simple left-to-right HMM.

4.3 Simulation and Tests of the Left-to-Right HMM

We present here the test process and the simulation results of the left-to-right HMM. The test parameters of the left-to-right HMM ($\lambda = (A, B, \pi)$) are generated randomly. The GMM parameters c_{jk}, μ_{jk}, Σ_{jk} (weights, mean vectors, and covariance matrices respectively for state j and mixture k), used for the computation of the emission probabilities of the hidden states are generated in the same manner as the GMM parameters for the EM Clustering algorithm. The non-zero entries of the state transition matrix are chosen randomly from a uniform distribution. The functionality of the HMM is verified by generating predetermined sequence of hidden states and the corresponding set of observation sequences. After that, the parameters of the left-to-right HMM are trained by the Baum-Welch algorithm, using the generated observation data as training data. After the left-to-right HMM parameters have been estimated, the Viterbi Search algorithm generates the most likely hidden state sequence of the observation set, based on the trained HMM model. The performance of the Baum-Welch and Viterbi Search algorithms are assessed by comparing the histograms of the observations corresponding to the predetermined and the generated hidden states. The set of GMM parameters regarding the emission probabilities of the hidden states is also compared (by plotting the data points of a particular state, the means and the probability contours of the predetermined and learned GMM parameters). The final performance assessment of the Baum-Welch and Viterbi Search algorithms of the left-to-right HMM is the classification rate. The classification rate of an HMM model is determined by comparing the estimated set of hidden state sequences to the predetermined set of hidden state sequences. The correct classification rate of the trained left-to-right HMM model is defined by the percentage of the decoded hidden states (in a set of hidden state sequences) matching the predetermined hidden states.

After the generation of the left-to-right HMM parameters $\lambda = (A, c_{jk}, \mu_{jk}, \Sigma_{jk}, \pi)$, each of the predetermined hidden state sequence and its corresponding training observation sequences are generated by the following procedure:

1. **The state index j is initialized to 1 and the time index n is initialized to 1.**
2. **The unit interval [0 1] is partitioned according to c_j. Then a random number is generated on the interval [0 1] and the mixture k is selected according to the partition whose subinterval boundaries contain the random number.**

3. **The observation for time index n is generated from the Gaussian distribution having a mean vector μ_{jk} and a covariance matrix Σ_{jk}.**

4. **The state j is recorded as the predetermined hidden state at time index n.**

5. **The unit interval [0 1] is partitioned by selection of the partition boundaries from the transition vector a_{ji} for $1 \leq j \leq S$. A random number is generated from a uniform distribution in the unit interval [0 1]. The next state j is determined by choosing the state partition whose subinterval contains the generated random number.**

6. **Increment the time index n to $n + 1$.**

7. **If the time index n is greater then the maximum permissable time index (20) or the last state of the left-to-right HMM is repeated the maximum number of times (5), then stop. Otherwise go to step 2.**

In an effort to decrease the time complexity of the learning and recognition systems, independence of the individual dimensions in the feature vectors is assumed during the Baum-Welch training and Viterbi Search of the feature vector set. The set of 2 dimensional observation sequences of the first experiment is generated randomly according to the process outlined above for a left-to-right HMM having 3 states and 5 mixtures in each state. The two dimensions of the generated observation set are independent and thus have a covariance matrix with off-diagonal elements of zero. The histograms of the states are smoothed by an averaging filter in order to demonstrate the envelope of the probability of the observation values in each of the 3 hidden states. Table 4.10 lists the HMM configuration parameters, used for the generation of the test HMM models and their corresponding observation sets. The value "Speech Variance" in Table 4.10 indicates that the variance of each dimension in the observation set is equal to the variance of the corresponding dimension of the speech feature vector set (accumulated from the TIMIT database via the feature extraction system).

Figure 4.14 shows the histograms of the observations in each of the 3 hidden states for both the trained HMM model and the predetermined HMM model.

Table 4.10: Parameters for the first left-to-right HMM test simulation (results are in Figures 4.14 to 4.18)

Parameter	**Value**	**Description**
S	3	Number of HMM hidden states used.
M	5	Number of GMM mixtures for each of HMM hidden states.
N	1000	Number of left-to-right HMM test observation sequences.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	Speech	Maximum possible value that an element in a covariance matrix can take.

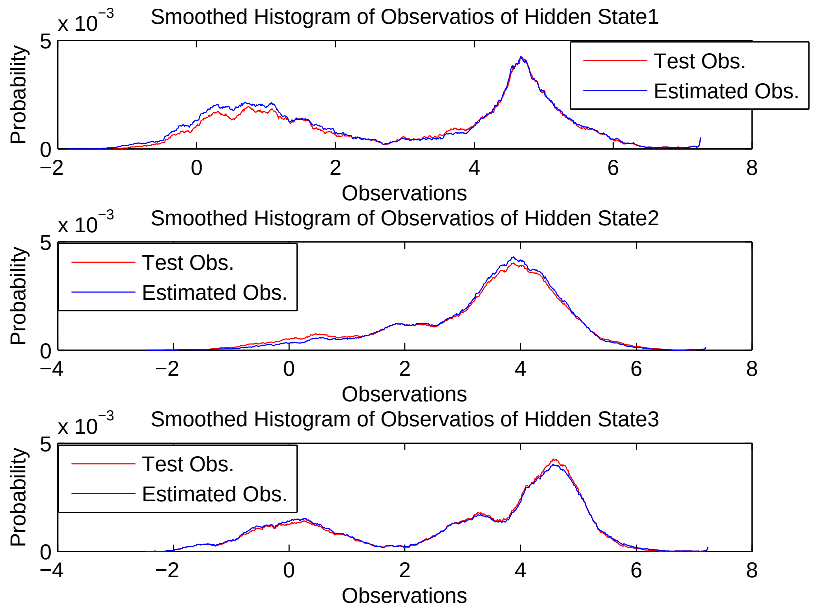

Figure 4.14: Observation histograms of the three hidden states of the test HMM (randomly generated) and trained HMM models. The observation set is 2 dimensional data with independent dimensions.

The three plots in Figure 4.14 correspond to the smoothed histograms of the first, second, and third hidden states respectively. As we see from Figure 4.14, the histograms of the observations of the predetermined set of hidden state sequences and the estimated set of hidden state sequences are fairly close to each other. This indicates that the 2 dimensional observations were, in fact, classified statistically correctly to their corresponding hidden states. The recognition rate of this trained left-to-right HMM model was 87%, which indicates a fairly low error rate (as expected from Figure 4.14). To further evaluate the performance of the Baum-Welch training and Viterbi Search algorithms we look at the plots of the observation data and the corresponding probability contours according to the predetermined and trained GMM parameters. Figures 4.15, 4.16, and 4.17 illustrate the predetermined and guessed observation data set and the probability contours for the first, second, and third hidden states respectively.

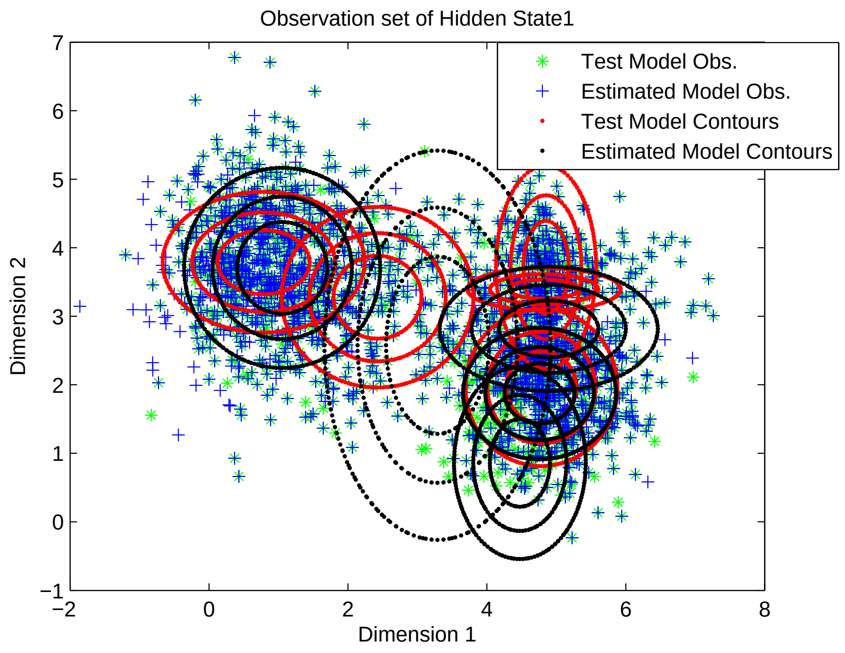

Figure 4.15: Test and estimated observations and probability contours of hidden state 1.

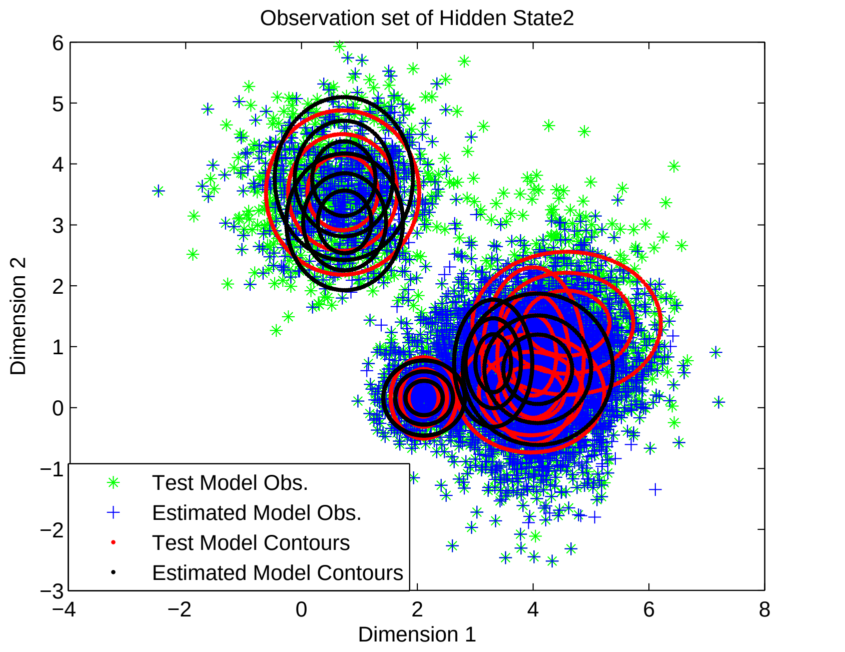

Figure 4.16: Test and estimated observations and probability contours of hidden state 2.

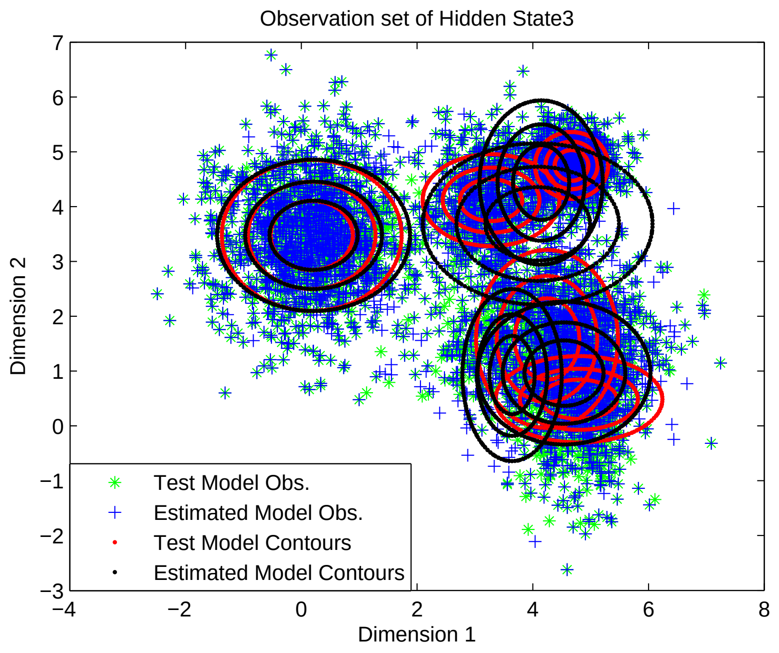

Figure 4.17: Test and estimated observations and probability contours of hidden state 3.

As observed in Figures 4.15, 4.16, and 4.17, most of the test and estimated observations for each of the three hidden states do overlap. Some of the probability contours of the test GMM and estimated GMM parameters have the same shape and are fairly close to each other, however some of the probability contours of the estimated GMM mixtures are within the observation set, but do not overlap with the test GMM mixture probability contours. This is most likely because the Baum-Welch re-estimation of the HMM parameters is an extension of the EM algorithm to the left-to-right HMM model and thus is guaranteed to converge to a local maximum (of the observation likelihood). Finally, we look at several of the individual hidden state sequences to verify the functionality of the Baum-Welch and Viterbi Search algorithms. Figure 4.18 shows the predetermined and estimated hidden states of four observation sequences.

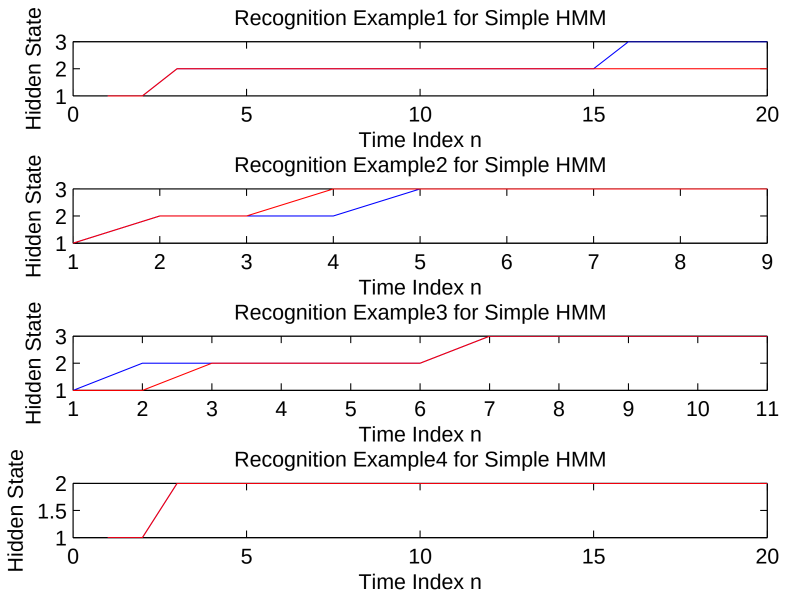

Figure 4.18: Predetermined and estimated hidden state sequences for four observation sequences.

The blue line in Figure 4.18 corresponds to the predetermined hidden sequence and the red line corresponds to the estimated hidden state sequence. As we see in Figure 4.18, all the estimated and predetermined hidden state sequences start with state one (because of the constraints on the state transition matrix A of the left-to-right HMM). Also note that the first and the last hidden state sequences in Figure 4.18 have 20 time indices, because the maximum allowable time index during the generation of the observation sequence is set to 20. Furthermore, the middle two plots of Figure 4.18 contain hidden state sequences whose length is less than 20, since the last state (state three) was repeated the maximum allowable number of times (5 times). Finally, as we see in Figure 4.18, most of the estimated hidden states correctly match the predetermined hidden states. Thus, Figures 4.14 to 4.18 demonstrate the performance of the Baum-Welch training and Viterbi Search algorithm results, given the observation sequence set generated from a 3 state, 5 GMM mixtures per hidden state, left-to-right HMM model.

In the next experimental setup we analyze the performance of the Baum-Welch algorithm and the Viterbi Search algorithm for a left-to-right HMM model that consists of 3 hidden states, 5 GMM mixtures per hidden state and a generated 25 dimensional observation sequence set. Each of the dimensions in the observation data set is assumed to be independent; as such the covariance matrixes of the predetermined GMM mixtures are diagonal matrixes. The purpose of this experiment is to evaluate the performance of the Baum-Welch and Viterbi Search algorithms for a 25 dimensional observation data set (since the speech feature vectors are also 25 dimensional). The randomly generated left-to-right HMM model is compared with the trained HMM model by comparing the smoothed histograms of the corresponding observations for each hidden state. Table 4.11 lists the configuration parameters used in this test simulation of the left-to-right HMM model training and recognition algorithms. Figure 4.19 shows the histograms of the predetermined and estimated observation data for the hidden states.

Table 4.11: Parameters for the second left-to-right HMM test simulation (results shown in Figure 4.19)

Parameter	**Value**	**Description**
S	3	Number of HMM hidden states used.
M	5	Number of GMM mixtures for each of HMM hidden states.
N	1000	Number of left-to-right HMM test observation sequences.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	25	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	Speech	Maximum possible value that an element in a covariance matrix can take.

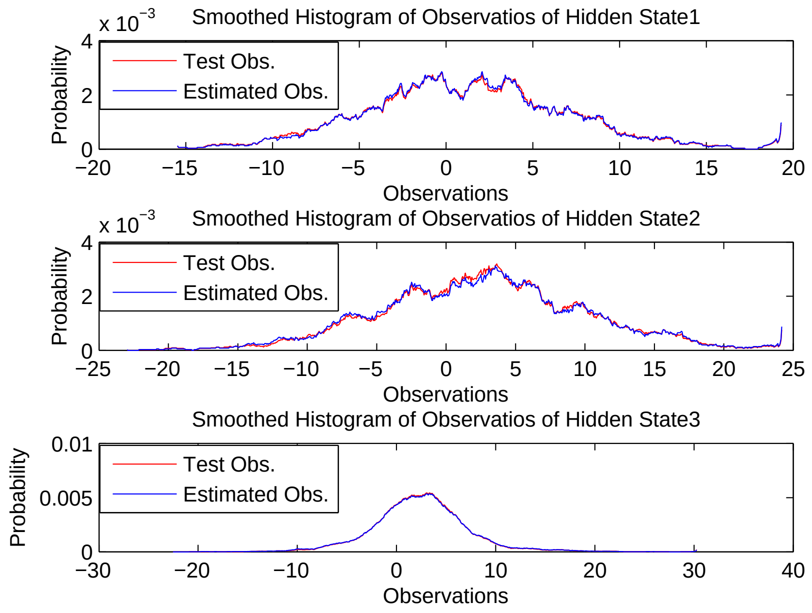

Figure 4.19: Observation histograms of the three hidden states of the test HMM (randomly generated) and trained HMM models. The observation set is 25 dimensional data with independent dimensions.

Similar to Figure 4.14, the three plots in Figure 4.19 correspond to the smoothed observation histograms of the first, second, and third hidden states. As we see in Figure 4.19, the histograms of the observations corresponding to the predetermined hidden states and the estimated hidden states match fairly closely. The evaluated classification rate of the observations to their hidden states was 95.21 % (as expected from Figure 4.19). Figure 4.19 also shows that there is no significant change in the performance of the Baum-Welch and Viterbi Search algorithms when the dimensionality of the observation data set is increased from 2 dimensional to 25 dimensional.

The final experimental setup analyzes the performance of the Baum-Welch and the Viterbi Search algorithms for a 25 dimensional observation sequence set, whose dimensions are not independent. However, the Baum-Welch and Viterbi Search algorithms assume independence between the dimensions of the observation sequence set. Thus the estimated GMM parameter set has diagonal covariance matrixes while the predetermined GMM parameter set has covariance matrices containing non-zero off-diagonal elements for this experiment. The predetermined HMM model is a 3 state left-to-right HMM model with 5 GMM mixtures for each hidden state. The performance of the Baum-Welch re-estimation and Viterbi Search algorithms is evaluated in the same manner as in the previous experiment (for the 25 dimensional observation sequences set). Figure 4.20 shows the histograms of the predetermined and estimated observation data corresponding to each of the three hidden states. Table 4.12 lists the experimental setup parameters used in the final test simulation of the left-to-right HMM model training and recognition algorithms.

Table 4.12: Parameters for the final left-to-right HMM test simulation (results shown in Figure 4.20)

Parameter	**Value**	**Description**
S	3	Number of HMM hidden states used.
M	5	Number of GMM mixtures for each of HMM hidden states.
N	1000	Number of left-to-right HMM test observation sequences.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	25	Dimensionality of the test observation data set.
Independent	false	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	Speech	Maximum possible value that an element in a covariance matrix can take.

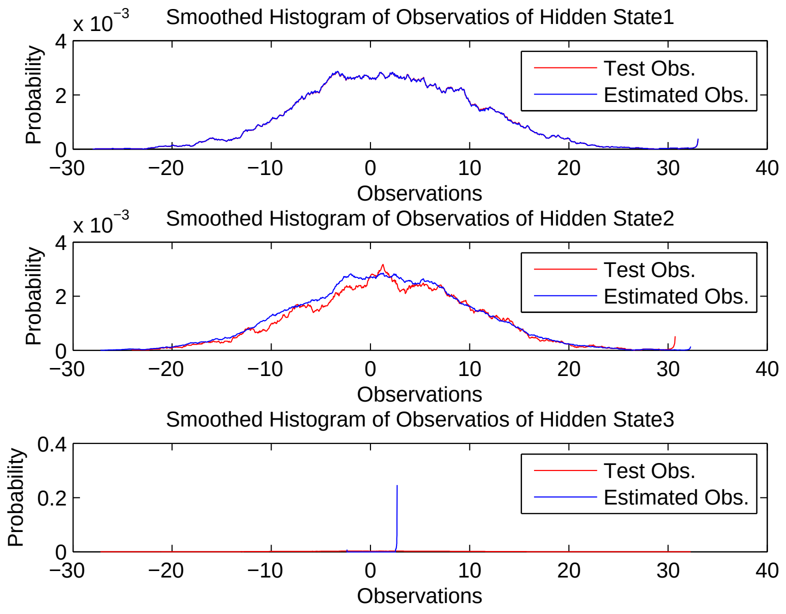

Figure 4.20: Observation histograms of the three hidden states of the test HMM (randomly generated) and trained HMM models. The observation set is 25 dimensional data with dependent dimensions.

As we see from Figure 4.20 there is a visible difference between the predetermined and estimated (smoothed) observation histograms corresponding to the three hidden states. The recorded classification rate of the hidden states of the trained left-to-right HMM model was 47.54 %. From Figure 4.20 and the classification rate we conclude that the violation of the independence assumption between the dimensions of the observation sequence can cause a significant degradation in the performance of both Baum-Welch and Viterbi Search algorithms. The next section presents simulation results of the performance of the Viterbi Search regarding the combination of the simple left-to-right HMM and the Segmental HMM topologies.

4.4 Simulation and Tests of the Segmental HMM

This section of the report mainly analyzes the performance of the Segmental Viterbi Search algorithm. Several tests are conducted to see the impact of the variation of some of the parameters in the Segmental Viterbi Search algorithm on its performance. The model and the observation sequence set of all of the tests performed for the validation of the Segmental Viterbi Search algorithm were generated in the same manner. For each experimental setup and test conducted, 10 left-to-right HMM models were randomly generated (in the same way as in Section 4.3) according to the specified number of states, GMM mixture components, dimensionality on the observation sequence sets and the independence assumptions of the observation data. The 10 left-to-right HMM models were then trained using the Baum-Welch re-estimation algorithm and the hidden state sequence for each of the observation sequence sets in each HMM model was estimated. The hidden sequence used for the Segmental Viterbi Search algorithm is denoted by the sequence of left-to-right HMM models. Thus there are 10 possible hidden states used for testing in the ergodic Segmental HMM. Each test observation sequence is generated in three steps. First, the sequence of hidden states (or the sequence of left-to-right HMM models) is determined. Second, an observation sequence for each HMM model in the hidden state sequence is generated. Finally, the observation sequences of all the hidden states (of left-to-right HMM models) are concatenated to form an observation sequence for testing the Segmental Viterbi Search algorithm. A set of such observation sequences comprises the observation data used for testing the Segmental Viterbi Search algorithm.

The goal of the Segmental Viterbi Search algorithm is to segment the observation sequence and to generate an optimum sequence of hidden states each of which represents one of 10 left-to-right HMM models, given the left-to-right HMM parameters trained by the Baum-Welch algorithm. The performance of the Segmental Viterbi Search algorithm is assessed by looking at the histogram of the correct classification rates of the observation sequences. A total of six separate experiments was conducted for performance evaluation of the Segmental Viterbi Search algorithm. During these six tests, the variance of the observation data set was varied, two different methods of choosing the optimum hidden state sequence were tested, two of the segmentation parameters (parameters used for determining the start and stop observation indices corresponding to the hidden states) were varied, and the performance of the Segmental Viterbi Search algorithm was tested with the independence assumption of the feature dimensions violated. These tests are described in detail below.

4.4.1 Test1: Small vs. Large Inter-Cluster Variance

The purpose of the first test is to compare the effects of observation data (with given GMM parameters for the hidden states) with small inter-cluster variance with a data set with high inter-cluster variance. Thus two different observation data sets where used for testing the effect of large observation data cluster overlap on the classification rate of the Segmental Viterbi Search. The first two dimensions of the observation data set are generated with unit variance (and zero covariance, since the two dimensions of the observation data are assumed to be independent). The corresponding variance of the second observation data was set to the variances of the first two dimensions of the speech feature vector set. These variances are 71.9501 and 17.7871 for the first and second dimensions (of the observation data) respectively. The means of the observation data are chosen randomly from the range of zero to the mean of the speech feature vectors for each dimension. Each of the two sets of generated observation sequences contain 600 observation sequences. For this experimental setup, the segment boundaries of the hidden states are known for the Segmental Viterbi Search algorithm. When the segment boundaries of the hidden states are known, the task of the Viterbi Search algorithm reduces to the classification of the within segment boundary observation sequence to the corresponding hidden model. When the segment boundaries of the hidden model sequence are not known, the two tasks of the Segmental Viterbi Search remain finding the start/stop observation indices of each hidden state (i.e. left-to-right HMM model) and estimating the corresponding sequence of hidden states. Table 4.13 lists the Segmental HMM parameters used for the training and testing of the Segmental Viterbi Search algorithm for this experiment, while Figure 4.21 illustrates the classification rates of the Segmental Viterbi Search algorithm for the two sets of observation data, with small and large variances respectively. The value listed as "Varies" for any of the parameters in the experimental setup tables indicates that the parameter is varied in this experiment and the results are recorded in the corresponding figures (listed in the table caption)

Table 4.13: Parameters for the first Segmental HMM test simulation (results shown in Figure 4.21)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	Varies	Maximum possible value that an element in a covariance matrix can take.
Method	Max	Method used for finding the optimum hidden state sequence.
2ρ	8	Range of observations used for possible hidden state start indexes.
τ	7 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	true	True if the segment boundaries are known. False otherwise.

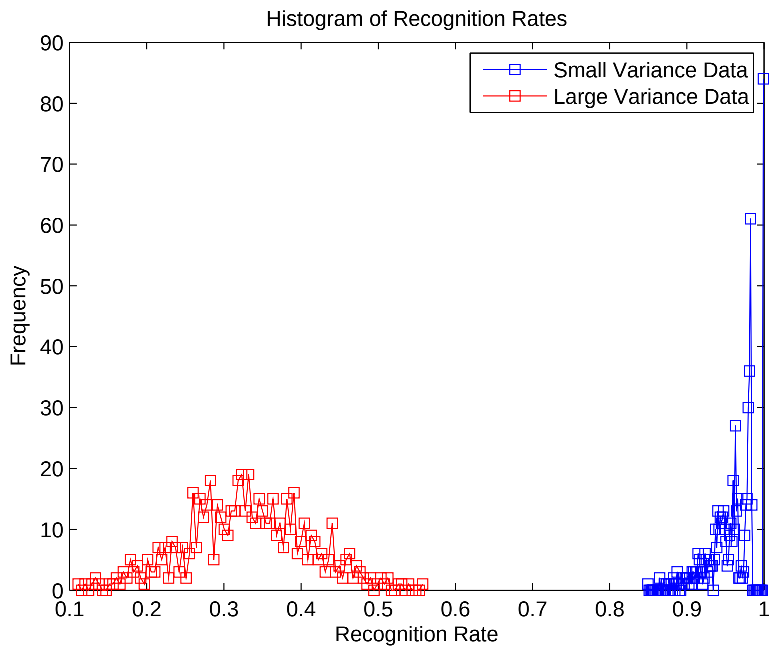

Figure 4.21: Comparison of the recognition results of the Segmental Viterbi Search algorithm when the observation data has small versus large variance.

As seen in Figure 4.21 the variance of the observation data has a large effect on the recognition rate of the Segmental Viterbi Search algorithm. The reported average classification rate for the Viterbi Search on the first and second observation data sets (with small and large inter-cluster variances respectively) was 96.16 % and 32.92 %. Since the variance of the second data set was the variance corresponding to dimension one and two of *all* the speech feature vectors, the speaker and speech variability yields a large variance. As we will see in the next section, several measures were taken to decrease both the speaker and speech variability of the phoneme observation data, however no significant improvements were recorded.

4.4.2 Test2: Given vs. Unknown Hidden State Segment Boundaries

The purpose of the second test is to compare the performance of the Segmental Viterbi Search algorithm under two different conditions. The recognition rate of the Segmental

Viterbi Search algorithm is compared for the case when the segment boundaries of the hidden states are given with the case where the segment boundaries of the hidden states are unknown. The set of observation sequences for both cases consists of 600 two dimensional observation sequences each. Table 4.14 lists all the parameters used in this experimental setup. Figure 4.22 shows the histograms of the 600 recognition rates (correct classification rates of each of the observation sequences) for both of these experimental setups.

Table 4.14: Parameters for the second Segmental HMM test simulation (results shown in Figure 4.22)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	1.5	Maximum possible value that an element in a covariance matrix can take.
Method	Max	Method used for finding the optimum hidden state sequence.
2ρ	8	Range of observations used for possible hidden state start indexes.
τ	7 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	Varies	True if the segment boundaries are known. False otherwise.

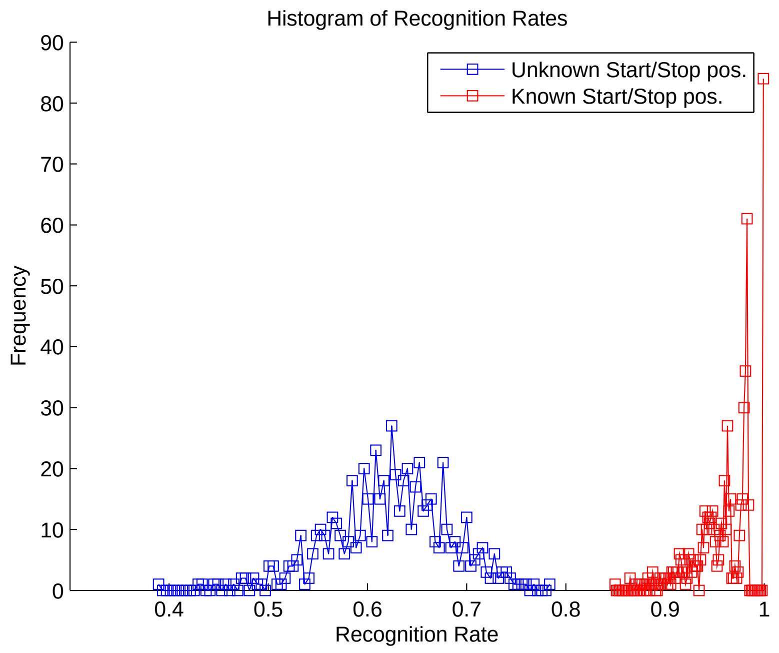

Figure 4.22: Comparison of the Segmental Viterbi Search algorithm results between the case where the segment boundaries are known versus being unknown.

As seen in Figure 4.22, there is a significant drop in the recognition rate when the start and stop observation indices of the hidden state models are not known. The average classification rate for the Segmental Viterbi Search algorithm with known start/stop indices was 96.16 % and the average classification rate with unknown hidden state segment boundaries was 62.23 %. From Figure 4.22 we deduce that the correct recognition of the hidden states segment boundaries plays a key role in the classification rate of the Segmental Viterbi Search algorithm. We proceed with the experimentation of the segment boundary constraint and threshold parameters that contribute to the determination of the start and stop observation indices of each of the hidden stat.

4.4.3 Test3: Variation of Range for Possible Observation Indices

In the third test setup we vary the range of the possible starting observation indices of each hidden state q_n, given the endpoint of the previous hidden state q_{n-1} (or 0 for $n = 1$). The range of the possible starting points of each hidden state is determined by the constant ρ and was illustrated in Figure 3.5. We examine the recognition rates of the Segmental Viterbi Search algorithm, given a set of 10 left-to-right HMM models, when varying the range of the possible hidden state starting points denoted by constant integer 2ρ. The configuration parameters for this experiment are listed in Table 4.15 and the experimental results are shown in Figure 4.23.

Table 4.15: Parameters for the third Segmental HMM test simulation (results shown in Figure 4.23)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	1.5	Maximum possible value that an element in a covariance matrix can take.
Method	2	Method used for finding the optimum hidden state sequence.
2ρ	Varies	Range of observations used for possible hidden state start indexes.
τ	7 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	false	True if the segment boundaries are known. False otherwise.

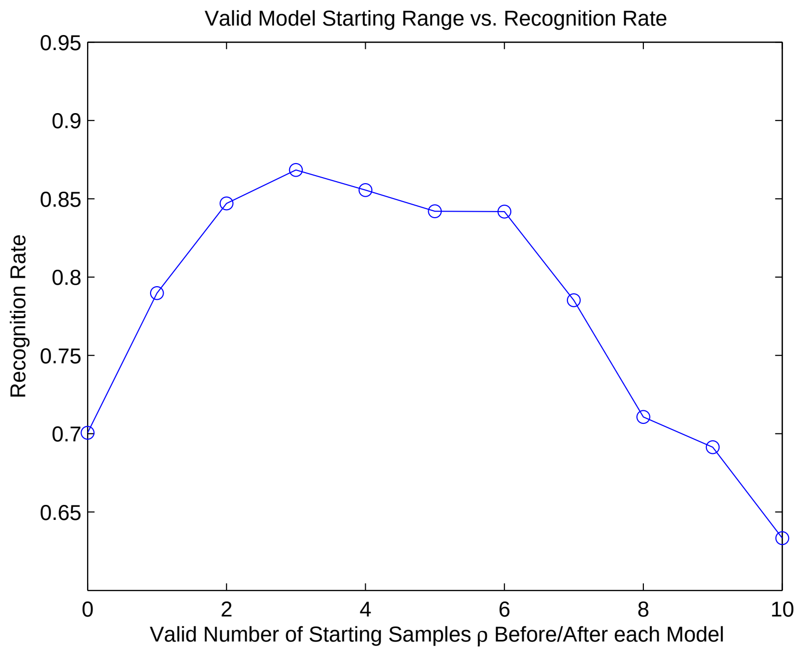

Figure 4.23: Recognition results for the Segmental Viterbi Search algorithm for varying range of observations allocated for the possible hidden state starting points (before and after the previous hidden state in the Segmental HMM).

Each of the points in Figure 4.23 corresponds to the mean of the recognition rates for 100 different observation sequences. As shown in Figure 4.23, the recognition rate peaks when the range of the permissable observation indices that can be chosen to represent the start observation of the hidden state q_n (or left-to-right HMM model) is three to four observations before and after the end of the previous hidden state q_{n-1}. Intuitively, as the range of the legal hidden state start observation indices increases, the chance of entirely skipping a hidden state-and/or inaccurately picking a starting observation index for a hidden state increases. On the other hand, as the range of permissable observation indices that can serve as a starting point of a hidden state decreases the robustness of the Segmental Viterbi Search algorithm also decreases. Thus, the concave plot in Figure 4.23 with a peak range for ρ is intuitively expected.

4.4.4 Test4: Effect of Threshold τ on the Recognition Rate

As mentioned in Section 3.3.2, the stopping point of the observation is computed by selecting the first observation index that has a deviation from the likelihood of the most likely left-to-right HMM hidden state sequence larger than a threshold percentage τ. This experimental setup demonstrates the effect of different values of the threshold τ on the correct classification rate of the Segmental Viterbi Search algorithm. The 10 left-to-right HMM model parameters and the corresponding observation data sets (100 observation sequences used for each test value of τ) were the same as in the previous experiment. The range of possible observation indices 2ρ for hidden state q_n given the last observation index of the previous hidden state q_{n-1} was set to 6. Table 4.16 lists the parameters of the Segmental HMM used for this simulation. Figure 4.23 illustrates the recognition rates of the Segmental Viterbi Decoding algorithm for various values of the threshold τ.

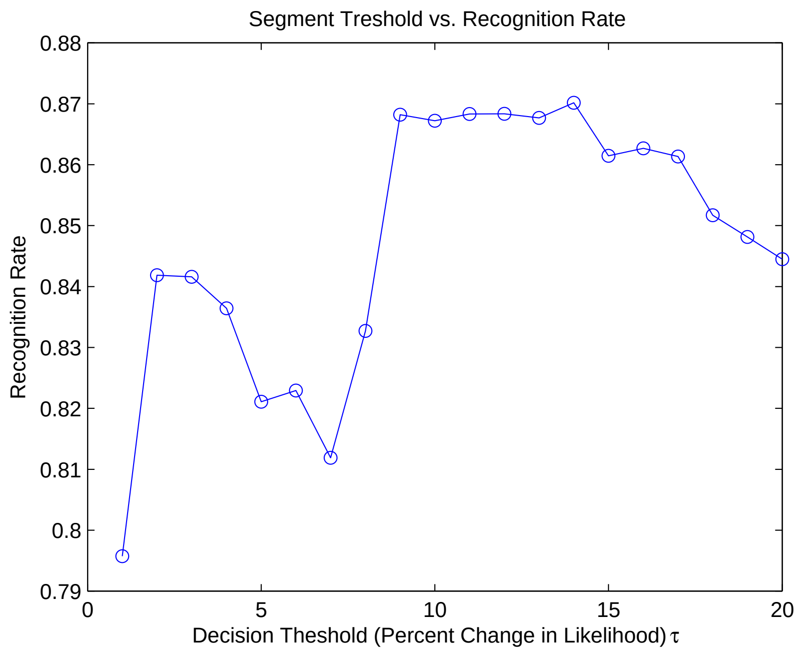

Figure 4.24: Recognition results for the Segmental Viterbi Search algorithm for varying likelihood thresholds for determination of the stop observation indices of the hidden states in a Segmental HMM.

Table 4.16: Parameters for the fourth Segmental HMM test simulation (results shown in Figure 4.24)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	1.5	Maximum possible value that an element in a covariance matrix can take.
Method	Max	Method used for finding the optimum hidden state sequence.
2ρ	6	Range of observations used for possible hidden state start indexes.
τ	Varies	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	false	True if the segment boundaries are known. False otherwise.

As for the previous test setup, each point in Figure 4.24 corresponds to the average recognition rate over 100 observation sequences. As we see in Figure 4.24, the recognition rate of the Segmental Viterbi Search algorithm is relatively large for values of τ ranging from 10 % to 14 %. For subsequent experiments therefore, a value of τ was chosen to be between 10 % to 14 %. This experiment concludes the simulations of all the parameters that affect the determination of the hidden state segment boundaries. In the next test setup two simulations are performed of two different methods for determining the sequence of hidden states that would yield to the highest classification rate in the phoneme recognition system.

4.4.5 Test5: Scaled vs. Max Viterbi Likelihood Method

As we have discussed in Section 3.3.2, there are two different methods that can be applied for the determination of the optimum hidden state sequence with the Segmental Viterbi Search algorithm. The Scaled Viterbi Likelihood method is the direct extension of the Viterbi Search algorithm to the Segmental Viterbi Search algorithm. Hence, the Scaled Viterbi Likelihood method determines the most likely hidden state sequence by selecting the Viterbi path from the set of possible paths Ψ with the highest scaled log-likelihood $\frac{\delta_n}{n}$. While this method results in the hidden state sequence with the highest normalized log-likelihood, simulations show that it is not the most practical approach for using the Segmental Viterbi Search algorithm for phoneme recognition. Intuitively this is due to the fact that a Viterbi path ending at a hidden state with a low final hidden state index has a higher chance of being the most likely Viterbi path than a path having a large final hidden state index. The Max Viterbi Likelihood method of selecting the final hidden state sequence, is to choose the Viterbi path that has the largest number of hidden states in the path maximizing the likelihood of the hidden state q_n at each hidden state index n. Since the recognition rate is determined by the percentage of correctly classified observation points, out of the total number of classified observation points, the classification rate of the Scaled Viterbi Likelihood method is expected to be higher than the classification rate of the Max Viterbi Likelihood method. However, simulations of the Scaled Viterbi Likelihood method show that only a few observations in an observation sequence are classified since, chances are that, the index of the final state yielding the most likely Viterbi path is small. On the other hand, most of the observations in the observation sequence are classified when the Max Viterbi Likelihood method is used (because of the nature of the objective function in the second hidden state sequence selection method). Table 4.17 lists the Segmental HMM parameter set used in this experiment, while Figure 4.25 shows a comparison of the recognition rates of the Segmental Viterbi Search algorithms for the two methods.

Table 4.17: Parameters for the fifth Segmental HMM test simulation (results are in Figures 4.25 to 4.27)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	2	Dimensionality of the test observation data set.
Independent	true	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	1.5	Maximum possible value that an element in a covariance matrix can take.
Method	Varies	Method used for finding the optimum hidden state sequence.
2ρ	6	Range of observations used for possible hidden state start indexes.
τ	12 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	false	True if the segment boundaries are known. False otherwise.

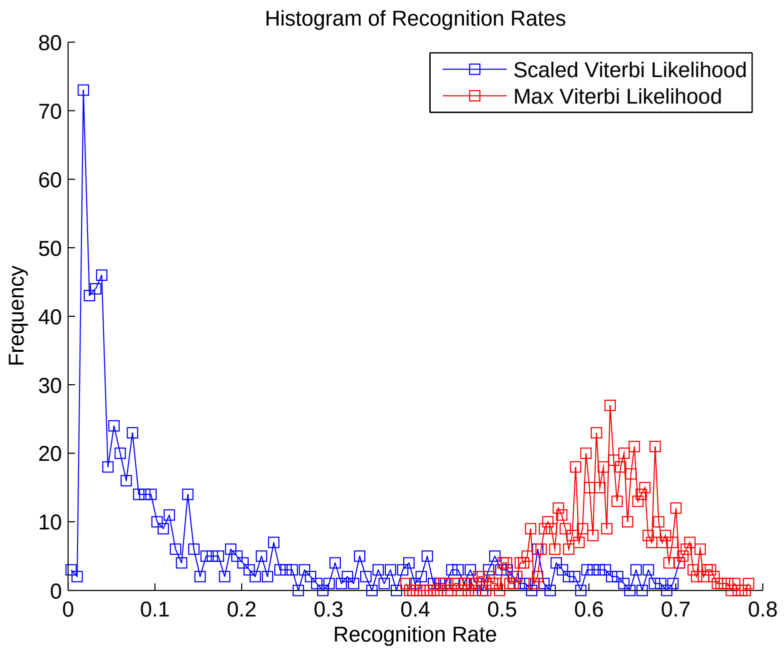

Figure 4.25: Comparison of Segmental Viterbi Search algorithm results for two methods of the hidden state sequence selection: The Scaled Viterbi Likelihood and the Max Viterbi Likelihood methods of selecting the Viterbi path as the hidden state sequence.

As seen in Figure 4.25 the classification rate of the observations to the corresponding hidden states when the Scaled Viterbi Likelihood method was used is indeed lower than the classification rate of the Segmental Viterbi Search algorithm when the Max Viterbi Likelihood method was used, as expected. The mean classification rate for the Segmental Viterbi Search algorithm using the Scaled Viterbi Likelihood and Max Viterbi Likelihood hidden state sequence selection methods are 15.88 % and 62.23 % respectively. Figures 4.26 and 4.27 present two different sample hidden state sequences and the corresponding estimated hidden state sequences produced by the Segmental Viterbi Search algorithm using the Scaled Viterbi Likelihood and Max Viterbi Likelihood methods respectively.

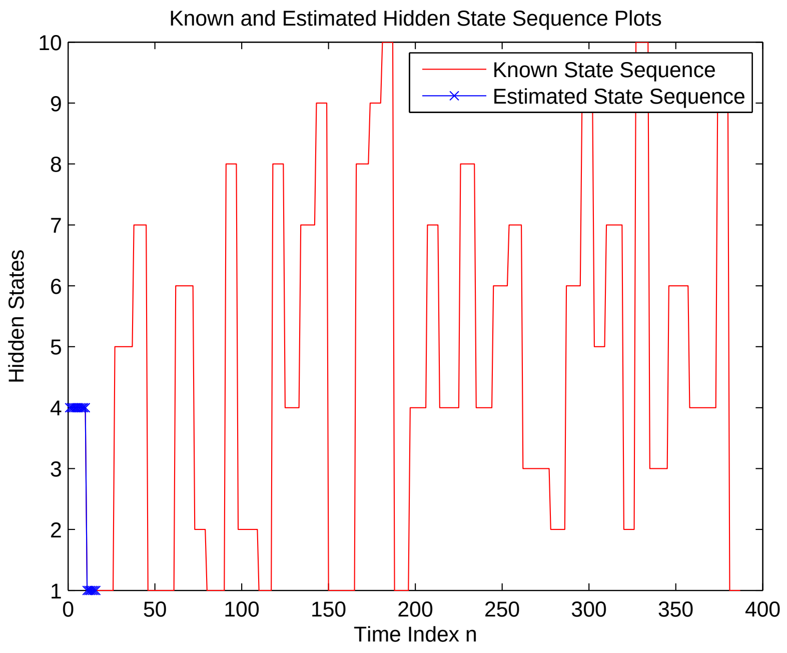

Figure 4.26: Known and estimated hidden state sequences when the Scaled Viterbi Likelihood method was used to choose the final hidden state sequence.

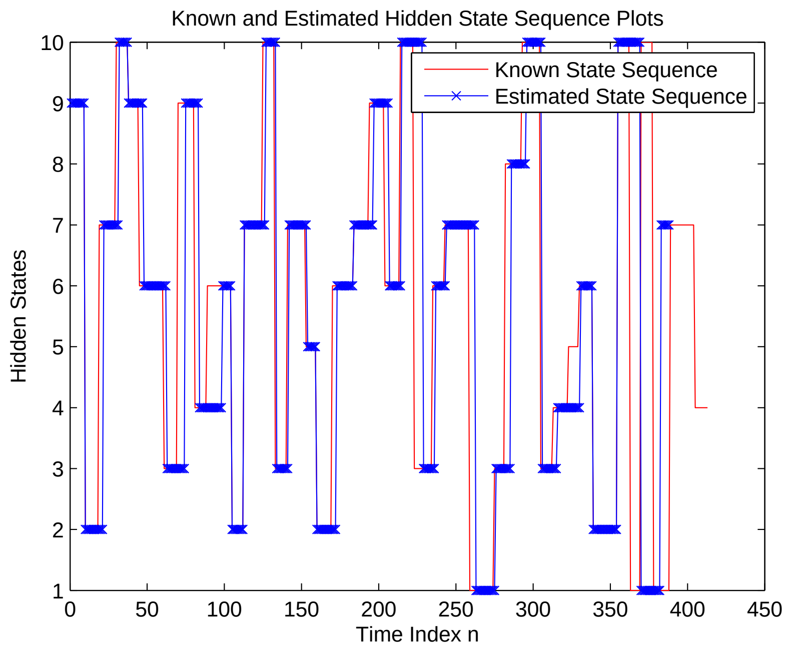

Figure 4.27: Known and estimated hidden state sequences when the Max Viterbi Likelihood method was used to choose the final hidden state sequence.

As seen in Figure 4.26, only 2 of the 40 hidden states are classified. Thus, this method is not practical for phoneme recognition (even though, 100 % of the points attempted for the recognition was classified correctly). On the other hand, the hidden state sequence in Figure 4.27 shows a higher recognition rate (than that in Figure 4.26), and almost all the observation points of the 40 hidden states (left-to-right HMM model observation sequences) in Figure 4.27 were classified. Thus Figures 4.26 and 4.27 show that the Max Viterbi Likelihood hidden state sequence selection method results in a higher phoneme recognition rate than the Scaled Viterbi Likelihood hidden state selection method. The phoneme recognition system incorporates the Max Viterbi Likelihood hidden state sequence selection into the Segmental Viterbi Search algorithm implementation.

4.4.6 Test6: Effect of Dependence Between Observation Dimensions

The final experiment tests the effect of violating the assumption of independence between the dimensions of the observation data set. The observation data set for this test experiment consists of 600 observation sequences, each of which has 25 dimensional randomly generated data. The set of GMM parameters pertaining to the 10 left-to-right HMM models have covariance matrices equal to the covariance matrix of the 25 dimensional speech feature vectors. Table 4.18 lists all the configuration parameters used in this experiment. Figure 4.28 illustrates the histogram of the recognition rates of the Segmental Viterbi Decoding algorithm when the assumption of independence between the observation data dimensions is violated.

Table 4.18: Parameters for the final Segmental HMM test simulation (results shown in Figure 4.28)

Parameter	**Value**	**Description**
S_{HMM}	10	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	1000	Number of left-to-right HMM test observation sequences.
N_{Seg}	600	Number of Segmental HMM test observation sequences.
N_{hs}	40	Number of hidden states in a segmental hidden state sequence.
L_{obs}	20	Maximum number of observations allowed in an observation sequence.
NUM_{LS}	5	Repeated last state threshold for a left-to-right HMM.
D	25	Dimensionality of the test observation data set.
Independent	false	True for independent observation data set dimensions. False otherwise.
Mean Threshold	5	Maximum possible value that an element in a mean vector can take.
Covariance Threshold	1.5	Maximum possible value that an element in a covariance matrix can take.
Method	Max	Method used for finding the optimum hidden state sequence.
2ρ	6	Range of observations used for possible hidden state start indexes.
τ	12 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	false	True if the segment boundaries are known. False otherwise.

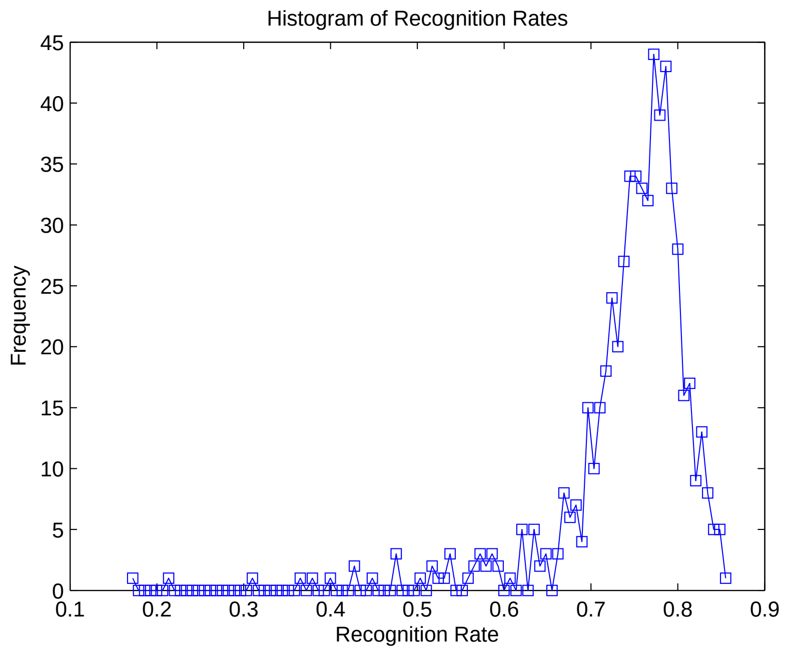

Figure 4.28: Histogram of the recognition rates of the Segmental Viterbi Search algorithm when the dimensions of the 25 dimensional observation set are dependent.

The mean recognition rate for this test simulation results shown in Figure 4.28 is 74.36 %. Since there was no significant drop in the classification rate, in comparison with the results obtained in the simulations of threshold variation (Test4 in Figure 4.24), we conclude that the violation of the independence assumption between dimensions of the observation data does not have a significant effect on the recognition rate of the Segmental Viterbi Search algorithm. In the next section we present the experiments performed on the speech data and a performance assessment of the automatic phoneme recognition system as a whole.

4.5 Simulation and Tests with the Speech Data

In this section we discuss the application of the algorithms presented in the previous chapter to speech data. The simulation analysis and tests of the implemented phoneme recognition system is also presented in this section. As we have seen in Test1 in Section 4.4.1, a large variance in the extracted feature vectors can have a big impact on the recognition rate of the phoneme recognition system. Most of the variation in the extracted speech feature vectors comes from the variation in the pronunciations of the phonemes in the speech utterance and the variation in the voices of the speakers recorded in the TIMIT database. Variation in the speech signal due to different pronunciations of the speech signal is called speech variability. The variations in the speech feature vectors due to variations in the speakers' voices is called speaker variability. Both of these variation sources can significantly degrade the performance of the phoneme recognition system. Thus two measures are taken to reduce the speech and speaker variabilities. From Tables 2.1 and 2.2 we see that there is a total of 60 phonetic symbols that are present in the speech database. Every speech utterance in the TIMIT database is hand segmented to a sequence of these phonetic symbols. To alleviate speaker variability of the extracted feature vectors, each of these feature vectors is categorized into one of two groups: phonetic symbols for male speakers and phonetic symbols for female speakers. Separation of the training data into males and females is known to reduce speaker variability, since there is a significant difference in the feature vectors (or in voices) of male speakers and female speakers [26]. Furthermore, speech variability is reduced by the introduction of context dependent phoneme states. The gathered feature vectors of 120 phonetic symbols (60 phonetic symbols for male speakers and 60 phonetic symbols for female speakers) are further divided into two subgroups: phonetic symbols that are preceded by a vowel and phonetic symbols that are preceded by a non-vowel. Table 2.1 contains a list of English vowels. Any phonetic symbol that is not in the list of vowels in Table 2.1 is considered to be a non-vowel. Thus the feature vectors of phonetic symbols are divided into four different groups: phonemes spoken by a male speaker and preceded by a vowel, phonemes spoken by a female speaker and preceded by a vowel, phonemes spoken by a male speaker and preceded by a non-vowel, phonemes spoken by a female speaker and preceded by a non-vowel, for a total of 240 phonetic symbols. Separation of the phonemes into phonemes that are preceded with a vowel or with a non-vowel is known to reduce speech variability in the feature vectors [11]. The number of feature vectors that have been extracted from the hand segmented speech utterances for each of the 240 phonetic symbols varies. Some of the phonetic symbols had fewer than 10 feature vectors extracted from the training set of the TIMIT speech database. Thus some of the 240 phonetic states do not have enough training data available to reasonably estimate the corresponding left-to-right HMM parameters (from small experiments we have deduced that at least 150 observation sequences are needed for effective parameter estimation). Thus for the implementation of the phoneme recognition system, the top 130 phonetic symbols were chosen containing 130 largest data sizes sorted from largest to smallest (5305 to 307 observation sequences respectively).

We present a set of simulations that scale the dimensionality of some of the 130 phonetic features down to see the underlying structure of the phoneme feature vectors and to compare different feature vectors for different phonemes. If the acoustic properties of two different phonemes differ significantly from each other (such as vowels and stops listed in Table 2.1) then we would expect to see a difference in the underlying structure of the phoneme data set for these two phonemes. The down-scaling of the dimensionality of the phonetic feature vector data set from 25 dimensional data to 2 dimensional data is performed by the MDS algorithm (discussed in Section 4.1). To visualize and see if there is a difference in the feature vectors between the two phonemes, the two 25 dimensional feature vector data sets are concatenated and scaled simultaneously. A $25 \times N_1$ and $25 \times N_2$ matrix results into $25 \times (N_1 + N_2)$ martix after concatenation. Figure 4.29 presents a comparison of the vowel "ax" and the stop "b", while Figure 4.30 visualizes the difference between the vowel "ax-h" and the nasal "m" (which are all listed in Table 2.1).

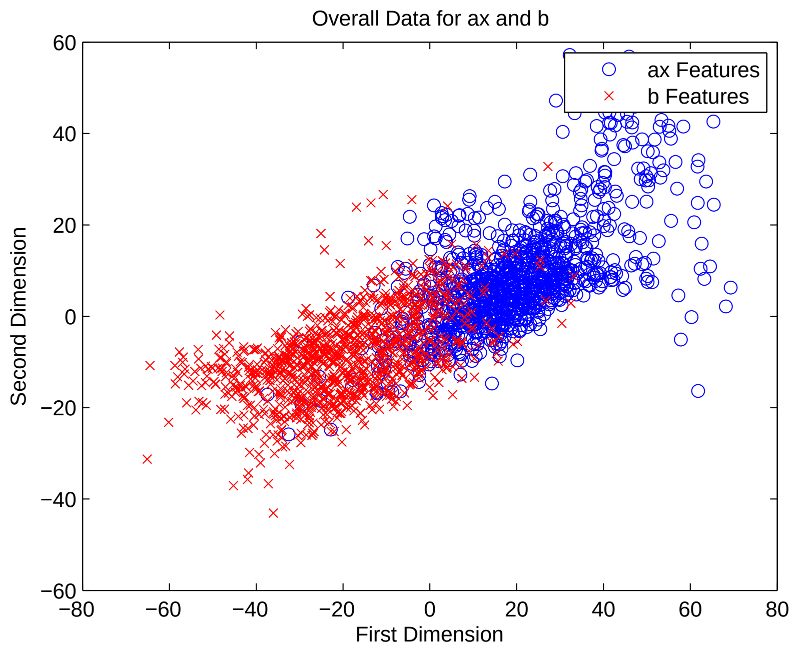

Figure 4.29: MDS Scaled feature vectors of the phonemes ax (a vowel) and b (a stop).

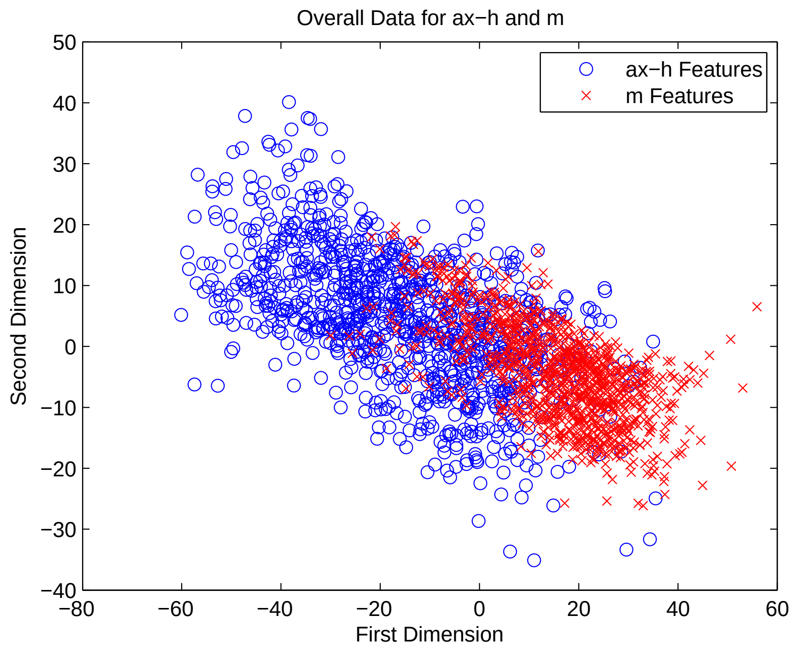

Figure 4.30: MDS Scaled feature vectors of the phonemes ax-h (a vowel) and m (a nasal).

As seen in Figures 4.29 and 4.30, even though there is some overlap, there is generally a clear difference between the scaled feature vectors of a vowel and the scaled feature vectors of a stop and a nasal. Hence, Figures 4.29 and 4.30 show the effectiveness of the feature extraction system and the choice of the extracted features. Once the features of all the 130 phonetic states have been extracted, the 130 left-to-right HMM model parameters have been estimated by the Baum-Welch algorithm for each of the corresponding phoneme states. These left-to-right HMM models are analogous to the 10 models used for testing the Segmental Viterbi Search algorithm. After the estimation of each phoneme state HMM model parameters, the recognition of these phoneme models is tested. For each of the 130 phoneme states, twenty observation sequences (or feature vector sequences) have been selected from the test set of the TIMIT database. For each of these observation sequences the HMM model yielding the highest log-likelihood of the Viterbi Search, taken over all the log-likelihoods computed from the Viterbi Searches of 130 phoneme HMM models, is labeled as the phonetic state associated with the observation sequence. The correct classification rate of the phoneme HMM model is determined by the percentage of the correctly classified phonetic states (out of 20 observation sequences). Table 4.19 lists the left-to-right HMM parameters used in the classification of the observation sequences to the corresponding phonetic states, while Figure 4.31 presents the histogram of the phonetic symbol classification rates.

Table 4.19: HMM parameters for the experiments of observation sequences to phonetic state classification. (results shown in Figure 4.31)

Parameter	**Value**	**Description**
S	3	Number of HMM hidden states used.
M	5	Number of GMM mixtures for each of HMM hidden states.
N	20	Number of left-to-right HMM test observation sequences.
D	25	Dimensionality of the test observation data set.

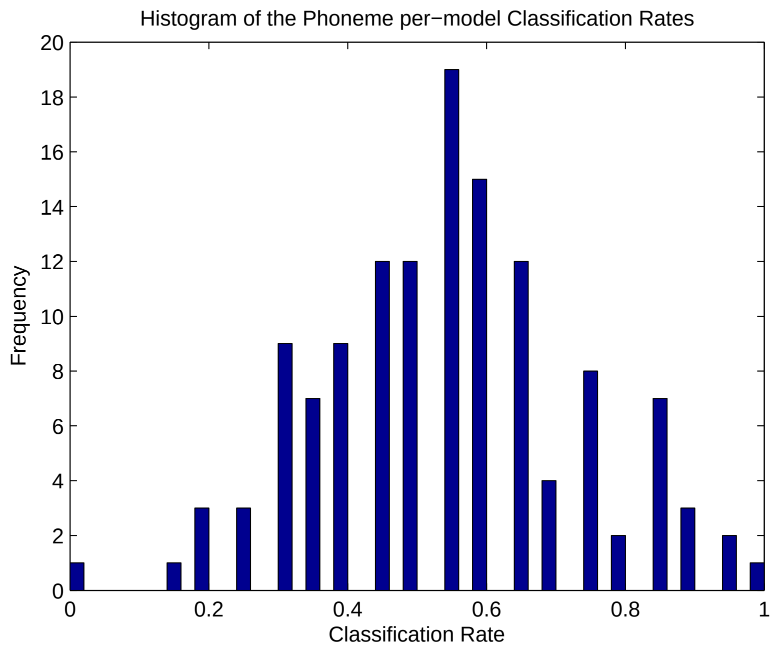

Figure 4.31: Histogram of the correct classification rates of the observation sequences to the corresponding (1 of the 60) phonetic symbols.

The average phonetic model classification shown in Figure 4.31 was 54.5 %. Similar classification rates of phoneme recognition systems have been reported [27] and [28]. T. Niesler in [27] performed a set of tests for monophone and triphone classification on a set of English, Afrikaans, and Xhosa phonemes, and reported about 42 % classification rate reported. J. Park in [28] has reported 51.79 % to 72.58 % phoneme classification rate for context independent phonemes. After the parameters of the 130 left-to-right HMM models have been estimated, the length statistic of each phonetic state (phoneme state length means and phoneme state length variances) is recorded (and is determined from the pre-segmented speech utterances in the TIMIT training data). Furthermore, the hidden state (or phoneme) transition matrix is also given by the percentage of transitions from a particular phoneme p_1 to another phoneme p_2 (from the total number of transitions from phoneme p_1). Thus the transition matrix is a 60×60 square matrix whose columns sum to unity (as described in Section 3.1). Once we have all the trained parameters for the second layer of recognition, the phoneme recognition system is tested by applying 160 speech utterances as input to the phoneme recognition system. The phoneme recognition system then processes these speech utterances and for each of the speech utterance observations the system outputs either one of the corresponding 60 phonetic states (listed in Tables 2.1 and 2.2), or a state -1 representing a before and after speech noise. The phoneme sequence recognition is performed by the Segmental Viterbi Search algorithm, using the segmentation boundary parameters maximizing the recognition rates ($\rho = 3$ and $\tau = 10$), as presented in Tests 3 and 4 in Section 4.4. Each of the classified observations in the output of the speech utterance is also compared to the hand-classified set of phonemes available in the test set of the TIMIT corpus. The end of the speech signal is set to the last observation index of the estimated hidden state sequences. Figure 4.32 shows a typical classification of the phoneme recognition system and the hand segmented hidden states in the TIMIT database for one of the 160 tested speech utterances.

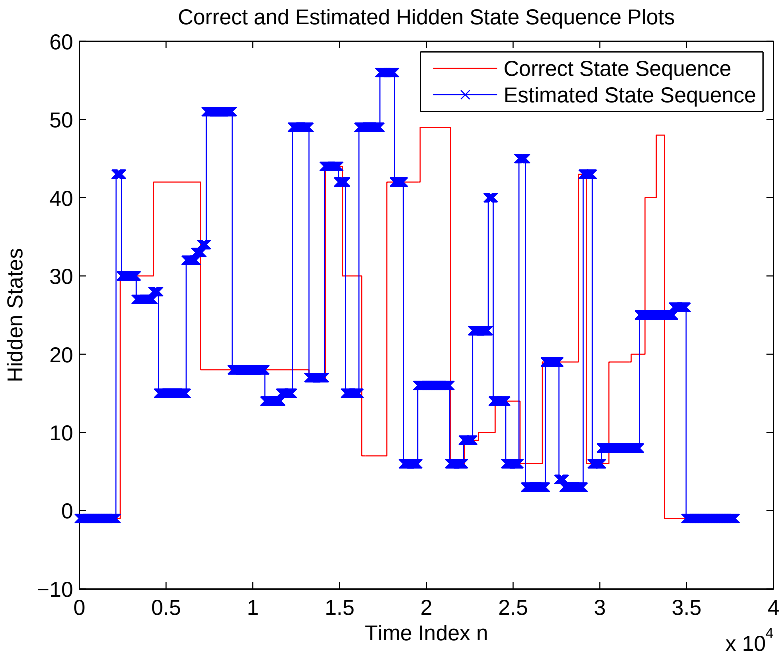

Figure 4.32: Example of an estimated and hand-segmented set of phoneme state sequences (estimated by the Segmental Viterbi Search algorithm) for a speech utterance.

As we see in Figure 4.32 the speech utterance starts and ends with the noise state (-1) for both the estimated and test hidden phoneme state sequences. This particular example has 34.76 % classification rate of the observation samples to the corresponding phoneme states. Table 4.20 lists the Segmental HMM parameters used in this experiment, while Figure 4.33 presents the histogram of the recognition rates of the phoneme recognition system tested for 160 speech utterances. In Table 4.20 the value "Varies" for the parameter "N" indicates that the number of observation sequences available for the training of each of the 130 left-to-right HMM parameters is not constant.

Table 4.20: Parameters for the Segmental HMM test simulation on the speech data (results shown in Figure 4.33)

Parameter	**Value**	**Description**
S_{HMM}	130	Number of left-to-right HMM models used.
S	3	Number of HMM hidden states used per left-to-right HMM.
M	5	Number of GMM mixtures for each left-to-right HMM state.
N	Varies	Number of left-to-right HMM test observation sequences.
D	25	Dimensionality of the test observation data set.
Method	Max	Method used for finding the optimum hidden state sequence.
2ρ	6	Range of observations used for possible hidden state start indexes.
τ	10 %	Change in Viterbi likelihood for hidden state stop index.
Known Bounds	false	True if the segment boundaries are known. False otherwise.

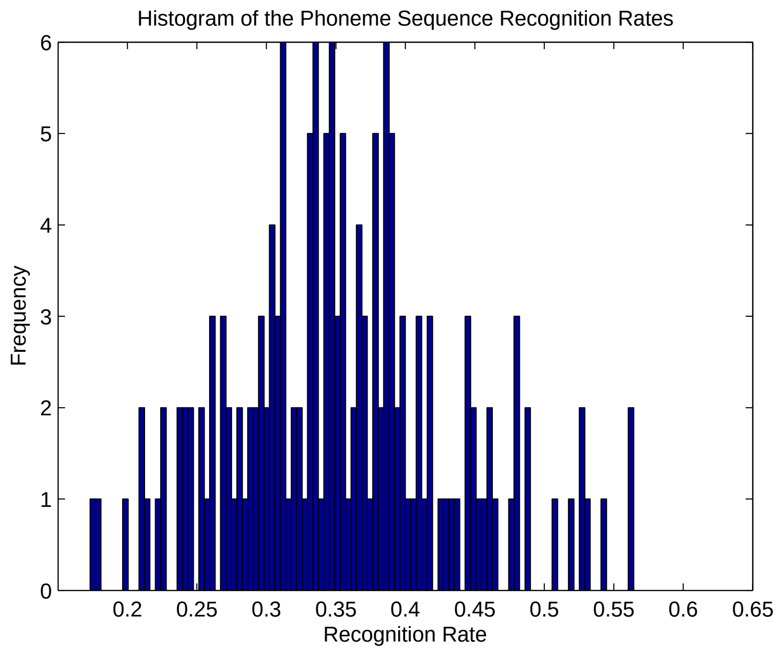

Figure 4.33: Histogram of the recognition rates of the phonetic symbol sequences (output of the phoneme recognition system) for 160 speech utterances.

The average recognition rate of the phoneme recognition system was 35.41 % with a minimum of 17.31 % and a maximum of 56.44 %. Phoneme sequence recognition systems performing joint recognition and segmentation task have been implemented and published in the literature [29, 30]. H. K. Kwan has implemented a phoneme recognition system applied to the task of continuous digit recognition using a hybrid combination of Artificial Neural Networks (ANN) and HMM [29]. The reported recognition rates ranged from 42.7 % to 91.7 %. The degradation in phoneme recognition rate is most likely due to the speaker and speech variabilities present in the phoneme state feature vectors.

In the next chapter conclusions are drawn from the algorithms implemented and tested in this thesis and several opportunities for improvements and future work.

Chapter 5

Conclusions and Future Work

The motivation for this research was to build a speaker independent phoneme recognition system that can be effectively incorporated in large vocabulary automatic speech recognition systems. The phoneme recognition system developed in this thesis incorporated phoneme transition information and phoneme length statistics by using the Segmental Hidden Markov model. In Chapter 1 a brief background was given on speech and speaker recognition and the application of the phoneme recognition system. In Chapter 2 the interleaved systems were presented that are involved in training and recognition of the phonetic data given in the TIMIT database of speech. In Chapter 3 a detailed discussion of the algorithms implemented for the training and testing of the phoneme recognition system was presented. The MFCC and Delta MFCC feature vectors were extracted from the training and test data sets of the TIMIT speech database. The Baum-Welch re-estimation algorithm was implemented for the estimation of the left-to-right HMM parameters for each of the phonetic symbols. The phoneme length and phoneme transition statistics were recorded. Finally, the phoneme recognition system used the accumulated training data and the implemented Segmental Viterbi Search algorithm (based on the Segmental Hidden Markov model) for the recognition of the phoneme string in the speech utterances of the TIMIT test set. A set of experiments and simulations that thoroughly test each of the algorithms described in Chapter 3 was presented in Chapter 4.

The recognition rate recorded for isolated phonemes (from a set of observation sequences pertaining to each phoneme) was 54.5 % (shown in Chapter 4, Section 4.5). The recognition rate for phoneme sequences in a continuous speech signal (after using the Segmental Viterbi Search algorithm) was 35.41 %. From the tests conducted on the Segmental Viterbi Search - as reported in Chapter 4 - we conclude that the highest factor in the degradation of the phoneme recognition results is the high speech and speaker variability that is present in the set of speech utterances in the TIMIT database. In the literature, several measures are taken to further reduce speech and speaker variability in the training and test data. One typical way of reducing the context variability in the speech signal is to incorporate a decision tree in the context dependent diphone or triphone phoneme models [11,31,32]. This method presents a way of grouping the extracted feature vectors into phonetic states (later to be converted into phoneme HMM models). This grouping is based on the type of phoneme preceding and following each phoneme. Thus each phoneme found in the training data set is grouped to a context dependent phoneme group based on a series of questions about the phonemes that precede and follow each phoneme. These questions are formed in decision tree(s) (for phonemes prior and following the phoneme being grouped). To compensate for the lack of training data available for the number of possible phoneme groupings applied, state tying is incorporated [33]. Another technique for reduction of speech and speaker variability is Successive State Splitting (SSS). The SSS algorithm is essentially a method of determining the context dependent subgroups of the feature vectors of each phoneme [34, 35]. Thus the SSS method and the preconstructed decision trees (according to the subgroups that yield maximum likelihood of the observation data) distribute the feature vectors of each phoneme into multiple different groups, which effectively have smaller speech variability (smaller than that of the phoneme features). These methods can be applied to the training of the phoneme left-to-right HMM models as future work.

There are also several alternative methods that have been researched for phoneme classification, such as Artificial Neural Networks (ANN) [36, 37] and Support Vector Machines (SVMs) [38]. These classification methods can also be integrated into the Segmental HMM of the phoneme recognition system presented in this thesis as future work.

Bibliography

[1] J. Joseph P. Campbell, “Speaker recognition: A tutorial,” *Proceedings of the IEEE*, vol. 85, pp. 1437 – 1462, 1997.

[2] C. J. Weinstein, “Opportunities for advanced speech processing in military computer-based systems,” *IEEE, Proceedings (ISSN 0018-9219)*, vol. 79, pp. p. 1626–1641, Nov. 1991.

[3] P. A. Srichai, “Implementation of a connected digit recognizer using continuous hidden markov modeling,” Master’s thesis, Virginia Polytechnic Institute and State University, September 1998.

[4] J.-L. Gauvain and L. Lamel, “Large-vocabulary continuous speech recognition: advances and applications,” *IEEE*, vol. 88, no. 8, pp. 1181 – 1200, 2000.

[5] J.-P. Haton, *Automatic Speech Analysis and Recognition.* NATO Scientific Affairs Division, 1981.

[6] P. Mermelstein, “Distance measures for speech recognition, psychological and instrumental,” *Academic, New York*, p. 374388, 1976.

[7] E. C. Bronson, M. R. Taaffet, and L. H. Jamieson, “A phoneme model of english speech for the design of a parallel speech understanding system,” in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference*, vol. 2, pp. 805 – 808, Purdue University, 1990.

[8] D. Cloarec, G.; Jouvet, "Modeling inter-speaker variability in speech recognition," *IEEE*, pp. 4529 – 4532, 2008.

[9] T. Imai, S. Sato, S. Homma, K. Onoe, and A. Kobayashi, "Online speech detection and dual-gender speech recognition for captioning broadcast news," *Oxford University Press*, vol. E90D, pp. 1286–1291, 2007.

[10] S. Sakti, K. Markov, S. Nakamura, and W. Minker, *Incorporating Knowledge Sources into Statistical Speech Recognition.* Springer Science+Business Media, 2009.

[11] J. J. O. S. J. Young and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," *Human Language Technology Conference*, vol. Acoustic modeling and robust CSR, pp. 307 – 312, 1994.

[12] T. T. Kristjansson, *Speech Recognition in Adverse Environments: A Probabilistic Aproach.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2002.

[13] J. H. Trausti Kristjansson, Hagai Attias, "Single microphone source separation using high resolution signal reconstruction," *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference*, vol. 2, pp. 817–820, 2004.

[14] V. Radova and Z. Svenda, "Speaker identification based on vector quantization," *Lecture Notes in Computer Science*, vol. Volume 1692/1999, p. 83, 1999.

[15] L. R. Rabiner and M. R. Sambur, "An algorithm for detecting the endpoints of isolated utterances," *The Bell System Technical Journal*, pp. 297–315, 1975.

[16] S. J. Melnikoff, S. F. Quigley, and M. J. Russell, "Speech recognition on an fpga using discrete and continuous hidden markov models," *Springer-Verlag*, vol. 2438, pp. 202 – 211, 2002.

[17] S. Nakagawa and K. Yamamoto, "Speech recognition using hidden markov models based on segmental statistics," *Wiley InterScience*, vol. 28, pp. 31–38, 1998.

[18] K. Achan, S. Roweis, and B. Frey, "A segmental hmm for speech waveforms," tech. rep., Department of Computer Science University of Toronto, 2004.

[19] Y.-S. Yun and Y.-H. Oh, "A segmental-feature hmm for speech pattern modeling," *Signal Processing Letters, IEEE*, vol. 7, pp. 135–137, 2000.

[20] M. Ostendorf, V. Digalakis, and O. Kimball, "From hmm's to segment models: a unified view of stochastic modeling for speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 4, pp. 360–378, 1996.

[21] M. Russell, "A segmental hmm for speech pattern modelling," *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference*, vol. 2, pp. 499–503, 1993.

[22] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. Vol. 39, pp. 1–38, 1977.

[23] D. Steinley and M. J. Brusco, "Initializing k-means batch clustering: A critical evaluation of several techniques," *IDEAS*, vol. 24, pp. 99–121, 2007.

[24] L. Wang, L. Tian, Y. Jia, and W. Han, "A hybrid algorithm for web document clustering based on frequent term sets and k-means," *Springer Berlin*, vol. 4537, pp. 198–203, 2007.

[25] J. de Leeuw and P. Mair, "Multidimensional scaling using majorization: Smacof in r," *Journal of Statistical Software*, vol. 31, no. 03, 2009.

[26] M. Zissman, "Language identification using phoneme recognition and phonotactic language modeling," *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference*, vol. 5, pp. 3503 – 3506, 1995.

[27] T. Niesler and P. Louw, "Comparative phonetic analysis and phoneme recognition for afrikaans, english and xhosa using the african speech technology telephone speech databases," *Pub Zone*, vol. 32, pp. 3–12, 2004.

[28] J. Park and H. Ko, "Real-time continuous phoneme recognition system using class-dependent tied-mixture hmm with hbt structure for speech-driven lip-sync," *IEEE Transactions on Multimedia*, vol. 10, pp. 1299 – 1306, 2008.

[29] H. Kwan, "Fuzzy neural network for phoneme sequence recognition," *Circuits and Systems, 2002 IEEE International Symposium*, vol. 2, pp. II–847 – II–850, 2002.

[30] J. Keshet, S. Bengio, D. Chazan, S. Shalev-Shwartz, and Y. Singer, "Discriminative kernel-based phoneme sequence recognition," tech. rep., Ecole Polytechnique, 2006.

[31] K. Beulen, E. Bransch, and H. Ney, "State tying for context dependent phoneme models," *ICASSP*, pp. 1179–1182, 1998.

[32] P. Banerjee, G. Garg, P. Mitra, and A. Basu, "Application of triphone clustering in acoustic modeling for continuous speech recognition in bengali," *IEEE*, pp. 1 – 4, 2008.

[33] Y. Liu and P. N. Fung, "Decision tree-based triphones are robust and practical for mandarian speech recognition," *Eurospeech 99*, vol. 6, pp. 895–898, 1999.

[34] J. Takami and S. Sagayama, "A successive state splitting algorithm for efficient allophone modeling," *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference*, vol. 1, pp. 573 – 576, 1992.

[35] H. Singer and M. Ostendorf, "Maximum likelihood successive state splitting," *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference*, vol. 2, pp. 601 – 604, 1996.

[36] D. Vassallo and E. Gatt, "Phoneme recognition using neural networks," *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference*, pp. 506 – 509, 2008.

[37] P. Brunet, Pandya, A.S., and C. Pinera, "Artificial neural networks for phoneme recognition," *Neural Networks, 1994. IEEE World Congress on Computational Intelligence*, vol. 7, pp. 4473 – 4478, 1994.

[38] J. Salomon, "Support vector machines for phoneme classification," Master's thesis, School of Artificial Intelligence Division of Informatics University of Edinburgh, 2001.

[39] J. Bilmes, "A gentle tutorial on the em algorithm including gaussian mixtures and baum-welch," *ICSI Technical Report TR-97-021*, vol. 21, 1997.

Appendix A

Derivations

We present here the derivation of the Expectation Maximization (EM) formulation for estimating the Maximum Likelihood (ML) parameter set of the Gaussian Mixture Model (GMM) parameters. The general EM algorithm for finding the ML parameter set is discussed first, followed by the application of the EM algorithm for the estimation of the GMM parameters (summerized in [39]). Assuming that the observation set $O = [o_1, o_2, \ldots, o_N]$ of N observations is generated from an underlying probability density function $P(O;\Theta)$ given a parameter set Θ we have:

$$P(O;\Theta) = \prod_{n=1}^{N} P(o_n;\Theta) \tag{A.1}$$

Equation (A.1) is valid because the observations in O are assumed to be independent and identically distributed. The goal of ML estimation is to find a parameter set Θ that maximizes $P(O;\Theta)$ in (A.1). Thus ML estimation finds a parameter set Θ^* such that,

$$\Theta^* = \operatorname*{argmax}_{\Theta} P(O;\Theta) \tag{A.2}$$

Most of the time finding Θ^* directly is a cumbersome task. The EM algorithm provides an iterative approach of finding Θ^* instead. The EM algorithm assumes a set of N hidden variables (not known directly from observations) $\Psi = [\psi_1, \psi_2, \ldots, \psi_N]$ such that,

$$P(O,\Psi;\Theta) = P(O|\Psi;\Theta)P(O;\Theta) \tag{A.3}$$

Let us define $R(\Theta, \Theta^{m-1})$ to be the expectation over the log likelihood of the hidden random variable set $\log(P(O, \Psi; \Theta))$ given the observation set O and the parameter estimate at the $(m-1)^{th}$ iteration (of the EM algorithm), given by:

$$R(\Theta, \Theta^{m-1}) = E_{\Psi}[\log(P(O, \Psi; \Theta))|O; \Theta^{m-1}] = \int_{\psi \in \Upsilon} \log(P(O, \psi; \Theta)) f(\psi|O; \Theta^{m-1}) d\psi \tag{A.4}$$

where $f(\psi|O; \Theta^{m-1})$ is the marginal distribution of the hidden random variables ψ given the observation set O and the parameter set of the $(m-1)^{th}$ iteration Θ^{m-1} and Υ is the space of values that ψ can take on [39]. The E-step (or the Expectation step) of the EM algorithm is the evaluation of $R(\Theta, \Theta^{m-1})$ in (A.4). The M-Step (or the Maximization step) of the EM algorithm is then to find a parameter Θ^m at iteration m that maximizes the quantity $R(\Theta, \Theta^{m-1})$. Thus,

$$\Theta^m = \underset{\Theta}{\operatorname{argmax}}\, R(\Theta, \Theta^{m-1}) \tag{A.5}$$

Equations (A.4) and (A.5) summarize the Expectation and the Maximization steps of the EM algorithm. We now proceed with the derivation of the EM algorithm for estimating the ML parameters of the GMM. Since, in this case, the observations are assumed to be generated from a GMM model, the probability of an observation $P(O; \Theta)$ (for a GMM parameter set Θ) is given by:

$$P(o_n; \Theta) = \sum_{k=1}^{K} c_k N(o_n; \mu_k, \Sigma_k) \tag{A.6}$$

where K is the number of components in the GMM mixture, $N(o_n; \mu_k, \Sigma_k)$ is the Normal probability given the mixture parameters μ_k and Σ_k for the k^{th} mixture component in the GMM, c_k is the corresponding weight of the k^{th} GMM mixture component and Θ is the GMM parameter set given by $\Theta = [c_1, c_2, \ldots, c_K, \mu_1, \mu_2, \ldots, \mu_K, \Sigma_1, \Sigma_2, \ldots, \Sigma_K]$. The Maximum Likelihood parameter set Θ^* maximizes the log likelihood of the observation set O, given the parameter set Θ and is given by:

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}} \log\left(\prod_{n=1}^{N} P(o_n;\Theta)\right) \tag{A.7}$$

$$= \underset{\Theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} c_k N(o_n;\mu_k,\Sigma_k)\right) \tag{A.8}$$

Finding Θ^* directly from (A.8) is difficult and cumbersome. Instead, the EM algorithm assumes the N hidden random variable set to be given by $\Psi = [\psi_1, \psi_2, \ldots, \psi_N]$ to denote the label of the mixture from which each of the observations o_n in the observation set O was generated. Thus ψ_n can be one of K labels $[1, 2, \ldots, K]$. The joint log likelihood $\log(P(O,\Psi;\Theta))$ of the observation set and the hidden variable set, given the GMM parameter set becomes

$$\log\left(P(O,\Psi;\Theta)\right) = \sum_{n=1}^{N} \log\left(P(\psi_n)P_{\psi_n}(o_n|\psi_n)\right) \tag{A.9}$$

$$= \sum_{n=1}^{N} \log\left(c_{\psi_n} N(o_n;\mu_{\psi_n},\Sigma_{\psi_n})\right) \tag{A.10}$$

The marginal probability $P(\psi_n|o_n;\Theta^{m-1})$ of the hidden variable set Ψ, given the observation set O and the GMM parameter set Θ^{m-1} for iteration $m-1$ is computed using Bayes rule, given by:

$$P(\psi_n|o_n;\Theta^{m-1}) = \frac{c_{\psi_n}^{m-1} N(o_n;\mu_{\psi_n}^{m-1},\Sigma_{\psi_n}^{m-1}))}{P(o_n;\Theta^{m-1})} \tag{A.11}$$

$$= \frac{c_{\psi_n}^{m-1} N(o_n;\mu_{\psi_n}^{m-1},\Sigma_{\psi_n}^{m-1}))}{\sum_{k=1}^{K} c_k^{m-1} N(o_n;\mu_k^{m-1},\Sigma_k^{m-1})} \tag{A.12}$$

and also

$$P(\Psi|O;\Theta^{m-1}) = \prod_{n=1}^{N} P(\psi_n|o_n;\Theta^{m-1}) \tag{A.13}$$

The expectation step of the EM algorithm is responsible for the computation of the quantity $R(\Theta, \Theta^{m-1})$ which is given by:

$$R(\Theta, \Theta^{m-1}) = E[\log(P(O, \Psi; \Theta))|O; \Theta^{m-1}] \tag{A.14}$$

$$= \sum_{\psi \in \Upsilon} \log(P(O, \psi; \Theta))P(\psi|O; \Theta^{m-1}) \tag{A.15}$$

$$= \sum_{\psi \in \Upsilon} \sum_{n=1}^{N} \log(c_{\psi_n} N(o_n; \mu_{\psi_n}, \Sigma_{\psi_n})) \prod_{j=1}^{N} P(\psi_j|o_j; \Theta^{m-1}) \tag{A.16}$$

$$= \sum_{\psi \in \Upsilon} \sum_{n=1}^{N} \sum_{k=1}^{K} \delta_{k,\psi_n} \log(c_k N(o_n; \mu_k, \Sigma_k)) \prod_{j=1}^{N} P(\psi_n = k|o_j; \Theta^{m-1}) \tag{A.17}$$

$$= \sum_{k=1}^{K} \sum_{n=1}^{N} \log(c_k N(o_n; \mu_k, \Sigma_k))P(\psi_n = k|o_n; \Theta^{m-1}) \tag{A.18}$$

Finally,

$$R(\Theta, \Theta^{m-1}) = \sum_{k=1}^{K} \sum_{n=1}^{N} \log(c_k)P(\psi_n = k|o_n; \Theta^{m-1}) + \sum_{k=1}^{K} \sum_{n=1}^{N} \log(N(o_n; \mu_k, \Sigma_k))P(\psi_n = k|o_n; \Theta^{m-1}) \tag{A.19}$$

During the Maximization step of the EM algorithm, the two terms of $R(\Theta, \Theta^{m-1})$ in (A.19) are maximized separately, since the first term only depends on the parameters c_k and the second term only depends on the parameters θ_k to be maximized. To find the update equation for c_k in the maximization step we introduce the Lagrange multiplier λ with a constraint $\sum_k c_k = 1$. The parameter set c_k is determined by solving the equation

$$\frac{\partial}{\partial c_k} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \log(c_k)P(\psi_n = k|o_n; \Theta^{m-1}) + \lambda(\sum_k c_k - 1) \right) = 0 \tag{A.20}$$

or

$$\sum_{k=1}^{K} \frac{1}{c_k} P(\psi_n = k|o_n; \Theta^{m-1}) + \lambda = 0 \tag{A.21}$$

If we sum both sides over k, we have $\lambda = -N$. Solving for c_k gives us:

$$c_k = \frac{1}{N} \sum_{n=1}^{N} P(\psi_n = k|o_n; \Theta^{m-1}) + \lambda = 0 \tag{A.22}$$

The second term for $R(\Theta, \Theta^{m-1})$ in (A.19) can be expanded to:

$$\sum_{k=1}^{K}\sum_{n=1}^{N} \log(N(o_n;\mu_k,\Sigma_k))P(\psi_n = k|o_n;\Theta^{m-1}) = \tag{A.23}$$

$$= \sum_{k=1}^{K}\sum_{n=1}^{N}(-\frac{1}{2}\log(|\Sigma_k|) - \frac{1}{2}(o_n-\mu_k)^T\Sigma_k^{-1}(o_n-\mu_k))p(k|o_n;\Theta^{m-1}) \tag{A.24}$$

Taking the derivative of the right hand side of (A.24) with respect to the mean μ_k and equating it to zero we get:

$$\sum_{n=1}^{N}\Sigma_k^{-1}(o_n-\mu_k)P(k|o_n;\Theta^{m-1}) = 0 \tag{A.25}$$

which is then solved for μ_k to find the updated means of the GMM mixtures. Thus μ_k is given by:

$$\mu_k = \frac{\sum_{n=1}^{N} o_n P(\psi_n = k|o_n;\Theta^{m-1})}{\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m-1})} \tag{A.26}$$

Similarly, we can find the covariance matrix Σ_k by setting the derivative of the right hand side of (A.24) with respect to Σ_k^{-1} to zero. Thus the covariance matrix is found by solving the following equation for Σ_k [39]:

$$\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m-1})(\Sigma_k - (o_n-\mu_k)^T(o_n-\mu_k)) = 0 \tag{A.27}$$

After solving (A.27) for Σ_k we get:

$$\Sigma_k = \frac{\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m-1})(o_n-\mu_k)^T(o_n-\mu_k)}{\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m-1})} \tag{A.28}$$

In summary, the update equations for the weights, mean vectors, and the covariance matrices in the Maximization step of the EM algorithm are given by:

$$\mu_k^{m+1} = \frac{\sum_{n=1}^{N} o_n P(\psi_n = k|o_n;\Theta^{m})}{\sum_{n=1}^{N} p(\psi_n = k|o_n;\Theta^{m})} \tag{A.29}$$

$$\Sigma_k^{m+1} = \frac{\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m})(o_n-\mu_k)^T(o_n-\mu_k)}{\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m})} \tag{A.30}$$

$$c_k^{m+1} = \frac{1}{N}\sum_{n=1}^{N} P(\psi_n = k|o_n;\Theta^{m}) \tag{A.31}$$