NTNU – Trondheim
Norwegian University of
Science and Technology

Machine Learning of Sub-Phonemic Units for Speech Recognition

Negar Olfati

Master of Science in Electronics
Submission date: February 2015
Supervisor: Torbjørn Svendsen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Machine Learning of Sub-Phonemic Units for Speech Recognition

Negar Olfati

Master of Science in Electronics

Submission date: February 2015

Supervisor: Torbjørn Svendsen

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem description

The current speech recognizers make use of phonemes or phones as the basic recognition unit, and have statistical description of phonemes modelled by acoustic models. For example, we have different models of /t/ when the sound is pronounced as in the word "star" compared to the word "butter". Phoneticians empirically employ the phoneme set of a language as a tool to provide an auditive description of that language. This means that phonemes are not necessarily the best units for automatic speech recognition.
This thesis aims to study methods to define an alternative set of units in speech, i.e. detecting the repetitive patterns and study the properties of the new units.

Assignment given: 15. September 2015
Supervisor: Professor Torbjørn Svendsen

To my dearest, Majid

To my lovely son, Kian

Abstract

This work is intended to explore the performance of a new set of acoustic model units in speech recognition. The acoustic models were built and evaluated from scratch in several steps: Feature extraction, acoustic detection and merging, acoustic segmentation of TIMIT corpus, clustering the segment representatives, assigning labels to each cluster and labelling the segments by cluster labels, and finally acoustic modeling. At the acoustic modeling phase, two experiments were investigated, using standard HMM structures and HTK toolkit; In the first experiment, the models were trained and evaluated by the annotated version of training data from TIMIT database in terms of cluster labels. In the second experiment, the time-aligned version of transcriptions was utilized to train acoustic models. Both experiments were carried out on four systems with 128, 256, 512 and 1024 units. Both single and mixture probability estimators were testified. In both experiments, the best results were achieved using GMMs with three-components for the 128 units system.

Acknowledgements

First of all, I would like to gratefully acknowledge Professor Torbjørn Svendsen for being supportive, patient and helpful at every phase during the work. Without his support, I couldn't have kept this work on the right track.

I also wish to express my sincere love to my family for encouraging me during the study, especially my husband Majid for being always next to me with his unconditional support. Without him, I could never have come this far. I wish I could thank him enough.

I am also thankful for the people behind the camera, those with the intellectual ownership of certain key ideas. They are all mentioned at the end of the report as reference.

Table of Contents

List of Abbreviations

AF	:	Articulatory Feature
ANN	:	Artificial Neural Network
ASR	:	Automatic Speech Recognition
DCT	:	Discrete Cosine Transform
FT	:	(Smola & S.V.N, 2008)Fourier Transform
HMM	:	Hidden Markov Model
HTK	:	The Hidden Markov Model Toolkit
MBE	:	Mel-bank Energy Coefficients
MMF	:	Master Macro File
MLF	:	Master Label File
MLP	:	Multi Layer Perceptron
MOA		Manner of Articulation
POA		Place of Articulation
SLP	:	Single Layer Perceptron
TIMIT	:	Texas Instrumental Massachusetts Institute of Technology

List of Figures

List of Tables

1 Introduction and Problem Setting

Speech is considered as the most natural way to communicate in everyday life. It has been made a lot of research to accomplish the process of mapping from speech signal to text by computers. This research field is called *speech recognition*. However, the task of recognizing any speech signal uttered by any person under any environmental condition is still a research challenge, and it is so far an easier task for human beings than for machines. Nevertheless, machines have been capable of working successfully in particular application areas, like dictation, augmentative communication, or call-routing.

1.1 What is machine learning?

Machine learning is the study of algorithms that learn the machine how to generalize the observed properties of an object to unobserved properties of that object [17]. The range of applications employing machine-learning techniques is large; some application areas are web page ranking, automatic translation of documents, face recognition, and last but not the least, speech recognition. In general, any area in which you need to make sense of data is a potential consumer of machine learning.

There are several different models available for learning a machine how to predict the unknown data by making use of the known data, such as classification by perceptrons, data clustering etc. In case of speech recognition, the known data is called the training set and the unknown data is called the testing set of a speech data set. In this work we will apply some of these methods.

There are two types of machine learning, depending on the availability/unavailability of the outputs. These are known as supervised and unsupervised approaches. In this project, a combination of these approaches is applied. For example, the technique used in training the neural networks can be categorized as supervised learning, while the designed k-means clustering algorithm is unsupervised [3].

1.2 Motivation of Research

One of the most essential factors in building automatic speech recognition systems (ASR) is to select the basic recognition unit. There are several criteria to be considered when selecting the recognition unit [7]:

- The unit should be *accurate* i.e. capable of modeling the acoustic realizations in different contexts.
- The unit should be *trainable*. For this purpose, enough training data should be available to estimate the unit parameters.
- The unit should be *generalizable* i.e. capable of creating higher level recognition units.

In small vocabulary tasks with specific set of words, the whole words are regarded as reasonable units. They are both accurate and trainable and there is no need for them to be generalized. However, in the case of large vocabulary ASRs, the use of whole words as basic units is a poor choice. In this case, accuracy and trainability would be a challenge since for "learning" each word to the system, several acoustic realizations would be required and hence the size of the required training data would be extremely large. In such systems, the phones may be a good alternative to construct the basic units for recognition. The problem with phones, however, is their accuracy, as they are not able to model the trajectories found in speech. In order to account for the acoustic variability, context-dependant sub-word models such as *biphones* and *triphones* are employed. A biphones model considers the adjacent phone on the left side of the phone being modelled, and a triphones model involves considering both its left and right phones.

In this project, in order to capture the coarticulatory effects of the speech signal, an alternative method is investigated. In the current approach, the units are built based on the articulatory features inherent in the signal. The advantage of these units over phones is that as they are built considering the articulatory features in the signal, they make the design of a universal language speech recognizer practical.

This thesis is conducted to investigate the behaviour and performance of these units through several experiments.

1.3 The Framework of This Project

A standard ASR system is constructed by five basic modules: speech parameterization, acoustic modeling, language modeling, decoding and the evaluation. Schematic diagram of a

typical ASR system is shown in Figure 1 [18]. The blue boxes represent the modules and the orange boxes represent the files associated with them.

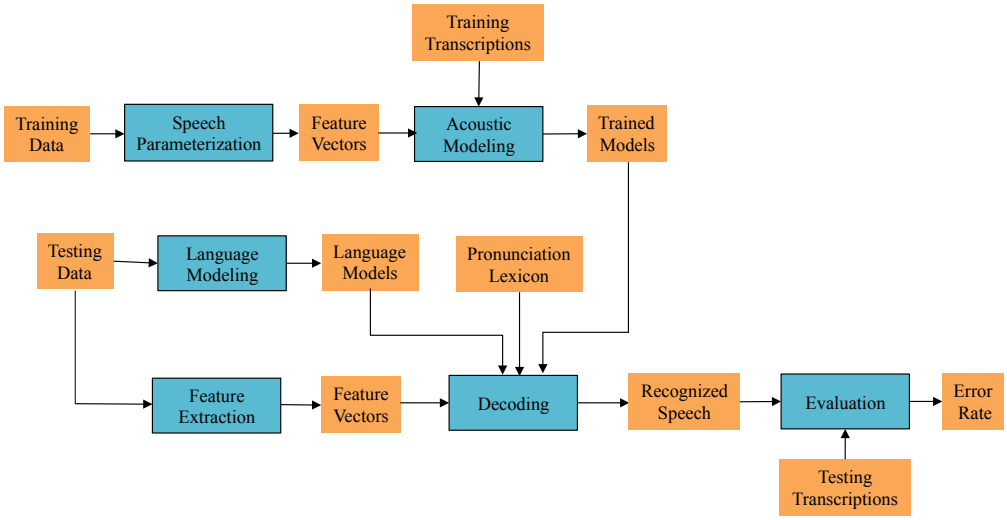


Figure 1: A Standard Speech Recognizer system

The system in this project is designed based on a segmental framework. In this manner, three extra steps compared to Figure 1 are involved in the work. These steps are depicted in Figure 2. They are shown in yellow and the files associated with them are in green. In this work, after the speech signals are parameterized, the inherent articulatory features existing in each frame are first detected and then merged in order to generate posterior supervectors. These vectors are then divided into segments each of which carrying a low degree of internal acoustic variation. For each segment, a representative vector found by averaging the posterior supervectors in that segment is identified. These segment representatives are then classified through a clustering algorithm (k-means). Then, the label corresponding to each cluster marks all segments of which their representatives belong to that cluster. These segmental units also referred to as sub-phonemic units, construct the basic units in our speech recognizer.

The articulatory detection phase of the system is performed by Artificial Neural Networks (ANNs), and the statistical modeling of the speech signals is done by Hidden Markov Models (HMMs). Such system is known as ANN/HMM hybrid speech recognition system. Furthermore, the current system is a continuous-speech speaker-independent ASR system. The term speaker-independent means that the system is designed to recognize anyone's speech, and the term continuous means that the words are pronounced without requiring any pause in between.

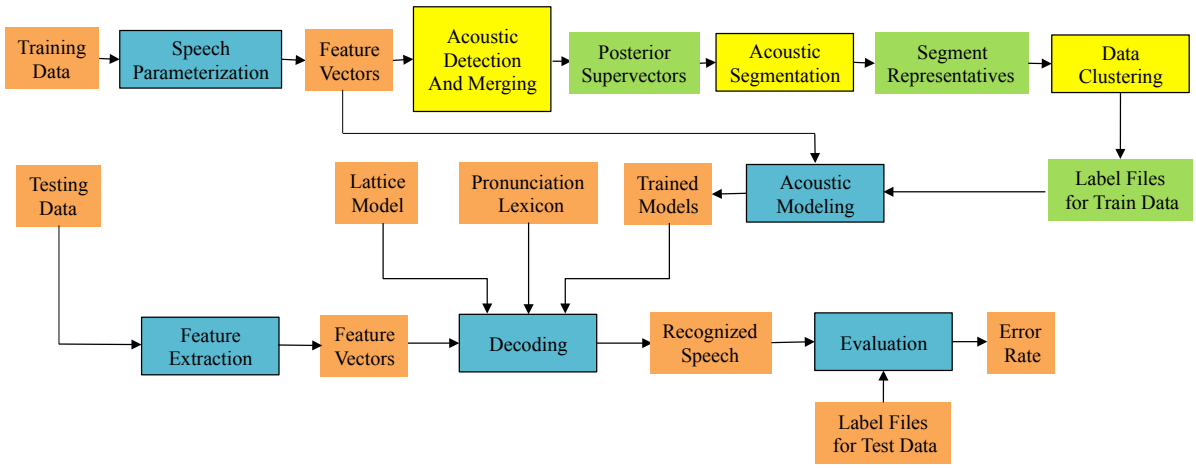


Figure 2: The overall framework of the system introduced in this work

It should be mentioned that due to time constraints, the designed system is just able to recognize in unit level. Therefore, the language modeling in Figure 1 is changed to lattice in Figure 2, which is a simple form of a language model. More complex language models are employed in higher levels of process, namely concatenating the models and building higher levels of speech units such as phones, syllables, and words, in order to decode what is said. All modules in Figure 2 are described thoroughly during the next chapters.

1.4 Preceding work

This project is connected to the specialization project performed during the spring semester 2014, in which by employing artificial neural networks, detection of AF features was investigated. In the current work, the obtained set of articulatory feature vectors from that project will be utilized.

1.5 How This Document is Organized

This document is partitioned into 9 chapters. The first chapter is the current chapter and provides an introduction to the work. Chapters 2 to 7 are structured in chronological order and cover all theoretical concepts required to understand the document. Chapter 8 is intended to report the experiments performed during the work, and discuss the obtained results. The last chapter includes a summary of the document and provides directions for future work. All the programming scripts are enclosed as Appendix. The scripts are commented carefully in order to enlighten the reader and they should be fairly easy to understand.

At the end of each chapter a brief summary is provided. Last but not the least, In order to understand this document it is assumed that the reader is familiar with the basic concepts regarding the probability theory and the Byes' rule, but not much else.

2 Speech Signal: Theory and Background

In this chapter an overview of the theoretical background regarding the production, representation and parameterization of speech signal is provided. The first section (2.1) represents the basic idea of how the speech is produced. At (2.2) the representation of speech in both time and frequency domain is described. The front-end analysis of the recognizer system employed in this work is introduced in (2.3). The chapter ends with a brief summary provided at (2.4).

2.1 Speech Production

Understanding the performance of the human speech production apparatus is essential in every speech recognition system, as all parts of the ASR system – from feature extraction to decoding – are inspired from the speech module in human beings. Therefore, in this section, we will provide a brief overview of the physiological structure of the speech production system. We will then describe how the speech sounds are produced through activating different organs in this system.

The speech production is started by firing control signals from the human brain to the speech apparatus system and thereby activating a special area in the system, while at the same time another control signal is sent towards the lungs in order to make them generate an air flow. The air stream is then passed through the larynx and the vocal folds, and thereby either voiced or voiceless sounds are produced. Next, the air stream enters the vocal tract. Here, due to the constriction of vocal tract in several places between vocal folds and mouth, the spectral characteristics of the speech signal are changed, and either vowels or consonants are created [10].

Figure 3 shows a simplified representation of physiological mechanism for speech production. Here the lungs are modelled by a source for producing the airflow. The position of the vocal cords will generate either quasi-periodic pulses or noise signal. A sequence of these speech sounds is then fed into a linear time-varying filter modeling the vocal tract, and hence a sequence of different speech sounds is produced.

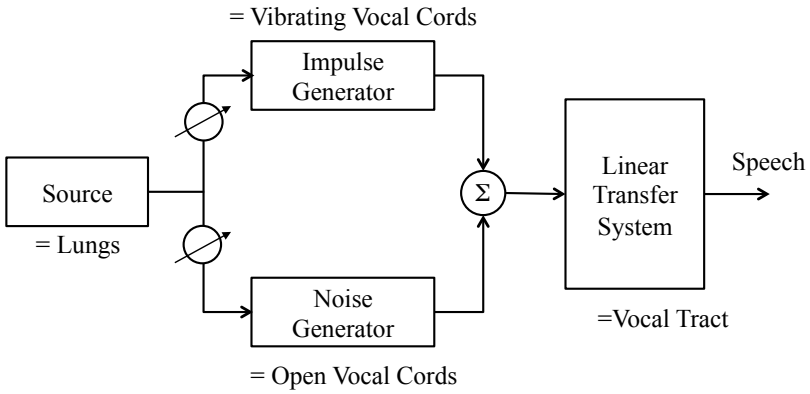


Figure 3: Schematic representation of the human speech production mechanism

2.1.1 Articulators and Articulatory Features

Articulators are the set of muscles with the ability to change the shape of the vocal tract and hence producing different sounds. Some of the main articulators are [10]:

- Pharynx: A tubular structure located between the larynx and the oral cavity.
- Velum: A soft structural tissue at the back of the mouth for controlling the airway.
- Hard palate: A smooth curved surface located between the oral and nasal cavities.
- Alveolar ridge: A rough surface with little ridges located between the upper teeth and the hard palate.
- Tongue: The most important articulator with greatest degree of interaction.
- Teeth: Located behind the lips
- Lips: The most visible articulators.

Both constriction of some of the articulators and interaction between them results in producing different speech sounds. These sounds are known as articulatory features (AFs), speech attributes, phonological features, articulatory cues, acoustic-phonetic features or articulatory-acoustic features in literature. AFs are regarded as the minimal units with the ability to be distinguished in a language. With such features, one is able to represent the overlapping nature of speech and model the contextual variation of signal [1,4].

The set of articulatory features used in this work are those represented in [16] by considering the following criteria:

- Manner of articulation (MOA): The degree of opening in the vocal tract when producing speech sounds.
- Place of articulation (POA): The relative interaction between articulators.
- Voicing: The vibrating degree of vocal folds.

These features together with their corresponding phoneme sets are provided in 22 groups in Table 1. According to this table, phones are built upward from a foundation of AFs [9]: each phone could be uniquely described as a supervector of 22 dimensions. Each component in the supervector signifies the presence or absence of a particular AF in that specific phone, and contributes to distinct that phone from the others. For instance, the existence of voiced feature accompanied with the absence of tense in /b/ will distinguish it from /p/, for which the reverse is true. One of the most important properties that these features own is being similar across languages. This makes the idea of designing a universal recognition system conceivable. Anther noticeable property of these features is their robustness against noise and cross-speaker variations [11]. These salient properties have motivated speech researchers to design recognition systems based on articulatory feature detection, known as detection-based ASRs.

Table 1: Articulatory feature set and relevant phonemes

Feature	Phoneme set	Feature	Phoneme set
Vowel	aa, ae, ah, aw, ay, eh, er, ey, ih, iy, ow, oy, uh, uw	Labial	b, f, m, p, v, w
Fricative	ch, dh, f, hh, jh, s, sh, th, v, z	Low	aa, ae, aw, ay, oy
Nasal	m, n, ng	Mid	ah, eh, ey, ow
Stop	b, d, dx, g, k, p, t	Retroflex	er, r
Approximant	l, r, w, y.	Velar	k, g, ng
Vocalic	aa, ae, ah, aw, ay, eh, er, ey, ih, iy, l, ow, oy, r, uh, uw, w, y.	Voiced	aa, ae, ah, aw, ay, b, d, dh, dx, eh,er, ey, g, ih, iy, jh, l, m, n, ng, ow, oy, r, uh, uw, v, w, y, z
High	ch, ih, iy, jh, sh, uh, uw, y	Round	aw, ow, oy, r, uh, uw, v, w, y
Coronal	d, dx, l, n, s, t, z	Tense	aa, ae, aw, ay, ch, ey, f, hh, iy, k, ow, oy, p, s, sh, t, th, uw
Dental	dh, th	Anterior	b, d, dx, dh, f, l, m, n, p, s, t, th, v, w, z.
Back	aa, ah, aw, ay, g, k, ow, oy, uh, uw	Glottal	hh
Continuant	aa, ae, ah, aw, ay, dh, eh, er, ey, f, ih, iy, l,ow, oy, r, s, sh, th ,uh, uw, v, w, y, z.	Silence	pau

A brief description for each feature is provided as follows [2,10].

- Anterior: Produced by an obstruction in front of the alveolar ridge.
- Continuant: Created when the oral tract is not entirely blocked.
- Coronal: Produced when the tongue blade is raised from its natural position, toward the teeth or the hard palate.
- Dental: Produced by inserting the tip of the tongue between the teeth.
- Fricative: Created by narrowing the vocal tract, causing turbulence as the air passes through it.
- Nasal: Produced by lowering the velum and allowing the air to pass through the nose.
- Stop: Generated when the air is not allowed to escape from the mouth.

- Approximant: Created by constricting the vocal tract slightly, but not so much as in fricatives
- Tense: Produced with a relatively longer duration and more articulatory interaction compared to other sounds.
- Glottal: Produced by passing the airflow through the open glottis and then out from the open mouth. The mouth is open because it is going to produce the next sound, which is always a vowel.
- Labial: Produced depend on the shape of the lips.
- Velar: Produced by raising the back part of the tongue to the soft palate.
- Retroflex: Produced with the tip of the tongue curled back toward the back of the alveolar ridge.
- Back: Produced when the tongue is retracted from its neutral position.
- High: Produced by raising the tongue above its neutral position.
- Low: Produced by positioning the tongue below its neutral position and far from the palate.
- Mid: Produced by locating the front of the tongue just above mid-height in the mouth.
- Round: Produced by making the lips rounded.
- Vocalic: Produced when the oral cavity is constricted by the same degree as when the vowels are produced.
- Vowel: Produced when the vocal cords are vibrating, but articulators not coming very close together.
- Voiced: Produced when the vocal folds are vibrating fully.
- Silence: presence of no sound is classified as silence.

2.2 Speech Representation

The raw speech signal can be represented in variant domains, such as time and frequency. Through these representations, one can extract valuable information from the signal and interpret its characteristics in a proper way.

In time domain, the waveform is displayed in two dimensions: intensity and time, and can be labelled via a three-state representation:

- Silence (S), in which the signal has low amplitude due to absence of sound;
- Unvoiced (U), in which the vocal cords are relaxed, generating the aperiodic signal with noise-like nature;
- Voiced (V), in which the vocal cords are vibrating periodically by passing the airflow

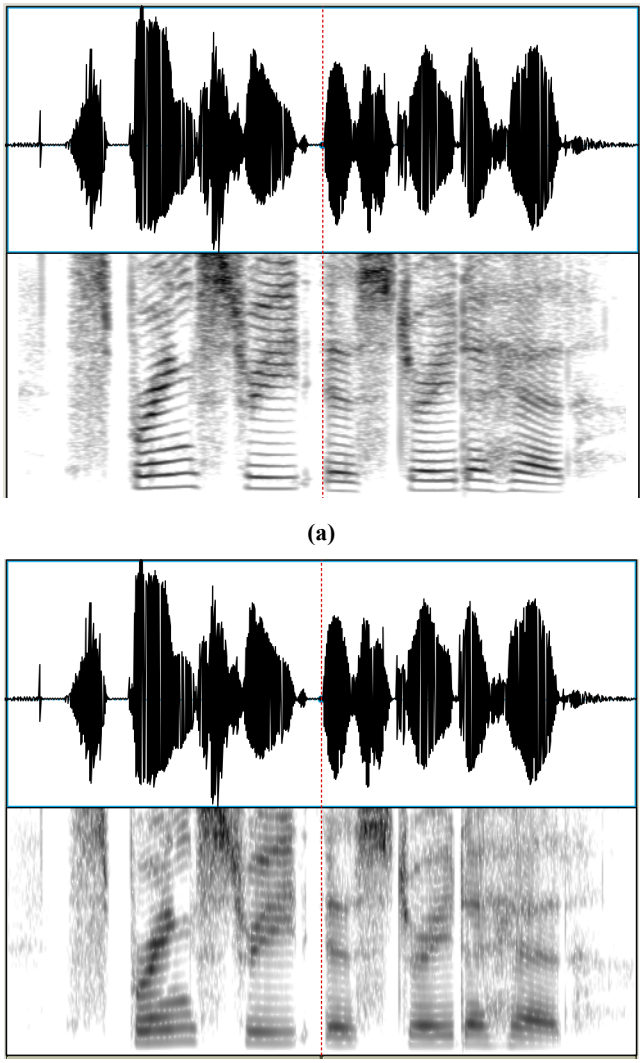
through them, producing quasi-periodic sound.

For time periods up to about 100 msec, the waveform clearly shows the slowly time varying nature of the speech. However, for longer time periods, the signal is non-stationary and the characteristics of speech signal will change resulting in different speech sounds being spoken. The time domain representation for "Spring Street is straight ahead." spoken by a male speaker is given in the upper row of Figure 4(a) and Figure 4(b).[1]

In frequency domain, the sound characteristics are interpreted via a spectral representation. The most popular form is the visual representation of the acoustic signal, known as *the sound* spectrogram. A spectrogram is a time-frequency-intensity representation of the speech signal, and can be seen as a tool to study the frequency change of the signal during the time.

In this type of representation, short time spectral analysis of all frames of the signal are represented as vertical lines, and the level of the grey scale represents the intensity (energy) content of the signal in different frequencies at different times. Depending on the size of the analysis window and the window shift, different levels of frequency/time resolution are obtained. A large window length (about 50 ms) along with a large window shift will result in high frequency and low time resolution. This type is known as *narrowband spectrogram*. Figure 4(a) shows a narrowband representation of the utterance "Spring Street is straight ahead".

[1] The diagrams in this section are plotted by the Praat speech toolkit. The speech file is adopted from TIMIT database.



(a)

(b)

Figure 4: (a) Narrowband and (b) Wideband Spectrograms for ”Spring Street is straight ahead.”

In the other hand, a short window (about 15 ms) length together with a short window shift will result in *broad-band spectrogram* with high temporal and low frequency resolution. In both cases, dark horizontal bands in the diagram are corresponding to the vocal tract resonance. Each component in the dark horizontal lines is called a *formant*, and is equivalent to a peak in the spectrum. Formants are the frequencies in which the signal carries the most acoustical energy.

Looking at these two diagrams, we see a smooth transition of formants over time. This gradually movement of formant frequencies, called *co-articulation*, is due to the influence of one speech sound on its neighbouring sounds.

2.3 Front-End Analysis

The front-end analysis in a speech recognizer plays a crucial role in the overall recognition performance of the system. This process deals with deriving the most meaningful information and discarding the irrelevant data from the speech signal.

As a preview of the section, Figure 5 shows an outline of the underlying sub-modules involved in the front-end analysis step in this work, namely mel-bank energy (MBE) feature extraction and split temporal context (STC) generation. The input to the front-end analysis module is the raw speech signal and the outputs are the STC vectors.

Figure 5: The front-end analysis step

2.3.1 MBE Feature Extraction

There are several methods available for parametrically representing a speech signal. Using these methods, the speech waveform will be transformed into a sequence of uncorrelated parameter vectors carrying the most essential information relevant to recognition process. The feature vectors employed in this work are the mel-bank energies (MBE). The steps regarding to the extraction of these features are shown in Figure 6 and described in brief in the following subsections.

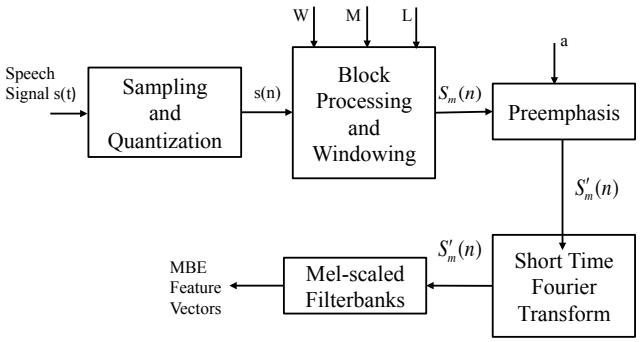


Figure 6: MBE Extraction Diagram

- Sampling and Quantization

The raw speech signal generated at the output of a microphone is first converted to a discrete-time sequence. This process is called *sampling*. The real-value number corresponding to amplitude of each sample is further converted to integer values, represented by a finite set of levels (bits). This is called *quantization*. Now, we have a digitized waveform with discrete-time, discrete-amplitude sequence of samples.[2]

- Block Processing and Windowing

In order to reduce the variability nature of the speech signal, it is divided into fixed-length overlapping segments. This is performed by multiplication of the speech signal by a proper window in time domain. A commonly used window (W) is *Hamming* window. This is a weighting window with the property to avoid discontinuity at the frame boundaries in time domain and correspondingly, reducing high frequency components in frequency domain. The window length (L) is typically set to 20-30 ms to ensure the speech stationarity inside the frames. The window shift (M) is typically set to 10 ms to cause overlap and ensure stationarity between frames.

- Pre-emphasis Filtering

Here, the speech frames are convolved with a first order FIR filter, known as pre-emphasis filter. This will increase the magnitude of higher frequencies with respect to lower frequencies in the speech signal, and results in a flattened spectrum.
The basic idea behind pre-emphasis filtering is to provide the best immunity to noise and measurement imperfections in next steps, compared to the non-emphasized signal [14, p.112].

- Short Time Fourier Transform

In this step, first a Fourier Transform (FT), and then the power spectrum for each frame is calculated. This process determines which frequencies are present in the speech sound.

- Mel-Scaled Filterbanks

In this step, the signal is passed through a bank of triangular bandpass filters inspired from the human auditory filtering and motivated by the fact that the hearing apparatus cannot resolve two closely located frequencies. The first filter is very narrow and indicates how much energy exists in low frequencies. The filters gradually get wider to model the less concerning nature of the hearing system about frequency variations. Then the energy that exists in various frequency ranges corresponding to each filter is computed.

The spectral-based feature vectors obtained after this final step are known as mel-bank energy (MBE) coefficients.

2.3.2 STC Feature Generation

In this step, for each frame to be processed, a group of MBE vectors corresponding to several frames at each side of that frame are passed through two modules: first half of a Hamming window and then temporal DCT. This is done in order to reduce the dimensionality and correlation among the coefficients. The procedure is shown in Figure 7. The special feature vector structure employed in this step is called the Split Temporal Context (STC) and results in vectors that model long temporal context of speech.

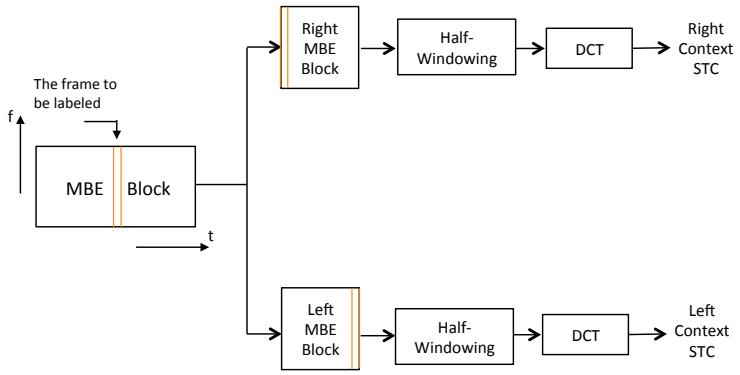


Figure 7: schematic diagram of STC structure

Recalling Figure 2, the first module is now completely described. In order to proceed to next module, namely acoustic detection and merging, it is necessary to introduce the theory of artificial neural networks and the basic idea of employing detectors. These are described in next chapter.

2.4 Summary

The chapter started by introducing the concept of creating sounds by human speech system, known as speech production. Then, several domains in which a speech signal can be represented were discussed. After that, all steps regarding the parameterization of raw speech in form of MBE features was described. The final section was about structuring the MBE features by using the STC generator, resulting in several times longer feature vectors with the ability to model long temporal context of the speech variations. These vectors construct the input data to the detection module, as to be described in the next chapter.

3 Artificial Neural Network Theory

This chapter deals with all necessary information in order to understand the second module in the speech recognition system depicted in Figure 2. First, a brief introduction regarding the artificial network theory is given in (3.1). Then the process of training neural networks is described in (3.2). After that, the acoustic detection procedure is treated in (3.3). This is followed by the acoustic merging technique described in (3.4). The chapter ends with a brief summary provided in (3.5).

3.1 Basics of neural networks

The human brains are made up of a combination of basic units called *neurons* sending electrical signals to each other. Natural neurons receive signals through synapses located on the dendrites or membrane of the neuron. Figure 8 shows a simplified view of a neuron. If the received signals are greater than a certain threshold, the neuron is activated and emits a signal through the axon. This signal might be sent to another synapse, and might activate other neurons [6].

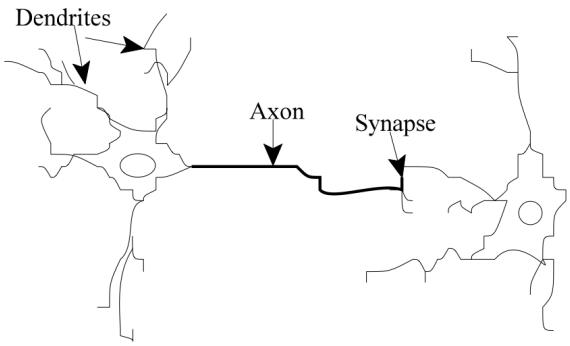


Figure 8: A basic view of a neuron unit

Inspired by how these neurons act in human brain, artificial neural networks (ANNs) have been designed. The schematic diagram of an artificial neural unit also known as perceptron

is shown in Figure 9. The model consists of input and output nodes, a summing junction and an activation function. In this model, 4 inputs receive the information from either the environment or other neurons. The inputs are then weighted by synaptic weight factors and summed up at the adder unit. Finally based on the activation function employed, the output is interpreted as either activated or deactivated. The activation function can be regarded as a model for how the human brain stores and processes information [8]. Typical activation functions are threshold function, piecewise linear function, sigmoid function and softmax function.

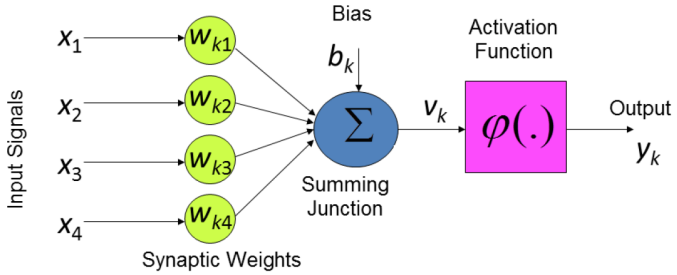


Figure 9: Modeling a perceptron

ANNs can be regarded as a valuable tool in artificial intelligence and also in data classification problems. Based on the number of hidden layers in their architecture, neural networks can be used in either linear or nonlinear classification problems: If there is no hidden layer, the network is called single layer perceptron (SLP) with linear classification property. This means that it can be employed in linearly separable classification problems. If the network consists of one or more hidden layers, it is called the multi layer perceptron (MLP) and is applicable in nonlinear classification problems. In this case, the distribution of data points in space is such that there is no linear solution to classify them.

3.2 Network Training

In order to train the networks, i.e. to determine the appropriate values for weight and bias variables, the *back propagation algorithm* is employed. This is a supervised learning technique in which both the input and the outputs are known to the algorithm. The process starts by feeding the input vectors to the network, and assigning random values to weights. At this time the output vector is computed. Then, the error defined by the difference between estimated and the original output values is calculated. The goal of the algorithm is to achieve

as minimum error as possible. In this regard, for each input vector, an iterative algorithm, starting at the output and going backward to the input is implemented to adapt the weights and the bias. The entire work is repeated until the weights converge to a final value [6].

At this point, we have all the materials to describe the second module of Figure 2, namely the acoustic detection and the merging process. These are described in the subsequent sections as follows.

3.3 Acoustic Detection

The resulted STC feature vectors, corresponding to left and right sides of the center frame obtained in section (2.3.2) will now be normalized to have zero mean and unit variance. These will then feed the input layer of detection module. This module consists of a detector bank with one detector for each AF. These detectors operate in parallel: for the available dataset, each detector is going to analyse the whole dataset independent of the other detectors. Figure 10 shows the detector bank architecture.

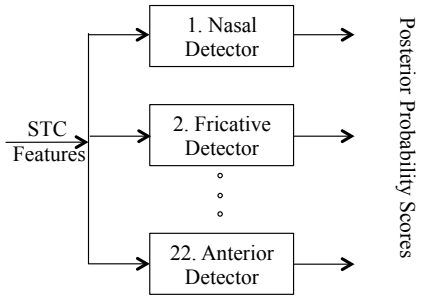


Figure 10: Schematic diagram of the detection module

Each detector in detection module has a hierarchical structure. That is, it consists of two context networks and one merger network, as shown in Figure 11. This is motivated by the fact that two parts of a phoneme may be detected independently – one considering the left, and the other considering the right context [13,15].

The context networks are MLPs consisting of one hidden layer of 500 nodes and output layer of 2 nodes. These two networks are trained to generate left and right context phonetic feature posteriors. Finally, the outputs of these context networks feed the merger network, with 4 nodes at the input layer, 500 nodes at the hidden layer and 2 nodes at the output. Outputs from this network generate a temporal sequence of posterior probabilities, each corresponding to

presence/absence of the attribute being studied in the speech frame.

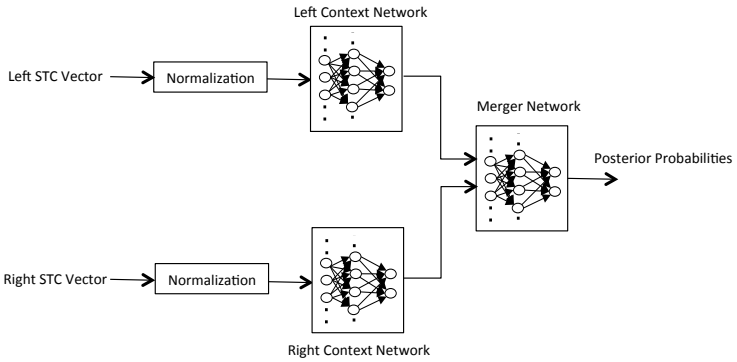


Figure 11: The Structure of One Detector in Acoustic Detection Module

As an illustration example, the posterior probabilities corresponding to the first 5 frames of the utterance *"He took a big swig of his drink."* With the TIMIT code mrtj0_si2032 generated at the output of four detectors – (a) Anterior, (b) Labial, (c) Velar and (d) Vid – are shown in Figure 12(a-d). Each row consists of four components: the two first components are the utterance number and the frame number, respectively. The last two components describe the occurrence/non-occurrence of that specific articulatory feature in the corresponding frame.

3.4 Acoustic Merging

After that all frames are classified by efficient parallel binary classifiers, output of these detectors are stacked in the form of a long vector of attribute posteriors.

Figure 12 (lower row) shows the posterior supervector corresponding to the first frame of the utterance mrtj0_si2032. This vector is of dimension 44. The first and second components in this representation are again related to the utterance and the frame number, respectively, and the last component indicates the manual phonemic label associated with that frame.

It is worth to mention that each two adjacent components in Figure 12(e) have complementary values. Therefore, just every other component in this vector suffices for existence/absence identification, and the other can be discarded without losing any essential information. Consequently, the dimension of posterior vectors generated after this step will be reduced to 22 during the experiments.

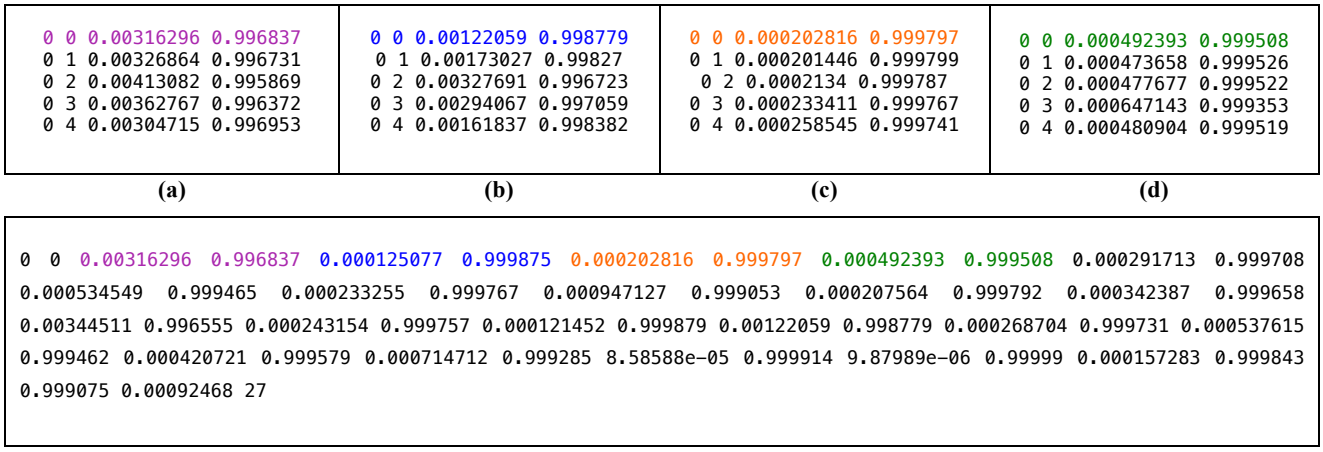


Figure 12 a-d) Anterior, Labial, Velar and Mid posterior probability outputs. e) The posterior vector

3.5 Summary

In this chapter, the theory of implementing neural networks in classification problems was described. Then, the idea behind AF detection by applying ANNs was explained. The detected features were then merged via a merging network, in order to generate posterior supervectors of dimension 44, which was reduced to 22 in a later process, to reduce the redundancy. With these posterior vectors at disposal, we will start describing the next module in Figure 2.

4 Acoustic Segmentation

This chapter deals with the theory regarding the third module in Figure 2, namely the acoustic segmentation. In this chapter, the idea behind segmentation is explained in (4.1). Then, the acoustic segmentation principle is described in (4.2). After that, the algorithm conducted to automatically segment speech into acoustical stationary segments is provided in (4.3). Then, the idea behind dynamic programming is described in (4.4). The chapter ends with a brief summary at (4.5).

4.1 Why Segmentation?

As described in section (1.2), the main motivation of this project is to create sub-phonemic recognition units. In this manner, we need to annotate the dataset in form of segmental units. For a relatively large vocabulary, the manual segmentation is impractical, because of the following:

- Manual segmentation implies interpreting spectrograms and aligning them with the continuous audio. This is quite time-consuming and tedious.
- The result is subjected to human errors and even two segmentation results of the same utterance may be inconsistent. [19]

For these reasons, automatic segmentation is regarded as a superior alternative to manual segmentation. The segmentation process is about to place boundaries along the speech signal, and thereby dividing the signal into variable length chunks. Two types of segmentation exist: phonemic and acoustic. Phonemic segmentation of the utterances is already available in this work. This is about placing the boundaries along the signal and specifying the phoneme boundaries in the signal. Acoustic segmentation is about placing the boundaries along the signal such that the resulted segments are acoustically homogeneous.

There are many algorithms proposed in the field of acoustic segmentation. In this work, the employed algorithm is similar to the one suggested in [19]. In the following, the principle of acoustic segmentation is described.

4.2 Acoustic Segmentation Principle

The 22-dimensional posterior vectors generated via acoustic merging module are now fed into the segmentation algorithm as input.

As mentioned above, in acoustic segmentation it is aimed to – as far as possible – divide the signal into acoustically homogeneous regions and end up with segments each of which corresponding to a phonetic event. In order to perform this, first of all some objective function is defined. Then the goal would be to optimize the fulfilment of that objective. The function in this work is selected to be the distortion error and the objective is to minimize this error. By that, each region of the signal in which the distortion error is less than a threshold is constructing a segment. The acoustic segmentation consists of two main components. They are described below.

4.2.1 The Representatives

As just mentioned, the main idea in segmentation is that each signal is segmented into stationary segments. This is achieved by representing all vectors in each individual segment by a fixed-dimensional vector called *representative* or *candidate*. The representative vector in this work, c_i, is selected to be the segmental centroid vector, that is

$$c_i = \frac{1}{N_i}\sum_{i=1}^{N_i} x_i[n] \qquad (1)$$

Where $x_i[n]$ and c_i are the posterior supervectors and the centroid vector in segment i, respectively, and N_i is the total number of frames in segment i.

4.2.2 The Distortion Measure

The distortion measure is generally defined as the difference between the original value and an estimation of the original value. In this work, the distortion measure is considered to be the sum of squared Euclidean distances between posteriori vectors belonging to a segment and the centroid vector of that segment.

For each segment i, the local distortion error d_i is found by:

$$d_i = \sum_{i=1}^{N_i} |x_i[n] - c_i|^2 \qquad (2)$$

Finally, the total distortion error D is found by:

$$D = \sum_{i=1}^{N_s} d_i \tag{3}$$

where N_s is the total number of segments in the utterance. The average distortion error would then be:

$$\bar{d} = \frac{1}{N_t} D \tag{4}$$

where N_t is the total number of frames in the spoken utterance.

4.3 The Acoustic Segmentation Algorithm

The algorithm is shown in Figure 13. It starts by dividing the signal in the "best" way into 2 segments; that is, to move the only required delimiter frame by frame and set it in all possible places, each of which giving a potential solution. For each potential solution, the local mean (c_i), the local distortion (d_i), and also the total distortion (D) are calculated. Then, the algorithm picks out the solution with the minimum total distortion as the best choice so far. For this solution, the average distortion ($\bar{d}$) is found and compared to a pre-defined threshold (ε). If its value is less than the threshold, the algorithm stops, meaning that we have reached the optimum result. Otherwise, the number of obtained segments in the utterance is compared to a pre-defined value, found by multiplication of over-segmentation factor (A) – a pre-defined constant corresponding to the maximum number of allowed segments – and the number of phonemes that exist in the utterance (#phn). If the number of segments is reached to that value, the algorithm stops. Otherwise, the number of segments is incremented by one and the same procedure applies. The algorithm is built upward until either it comes to a solution in which the average distortion is less than a pre-defined threshold, or a pre-defined number of segments is obtained.

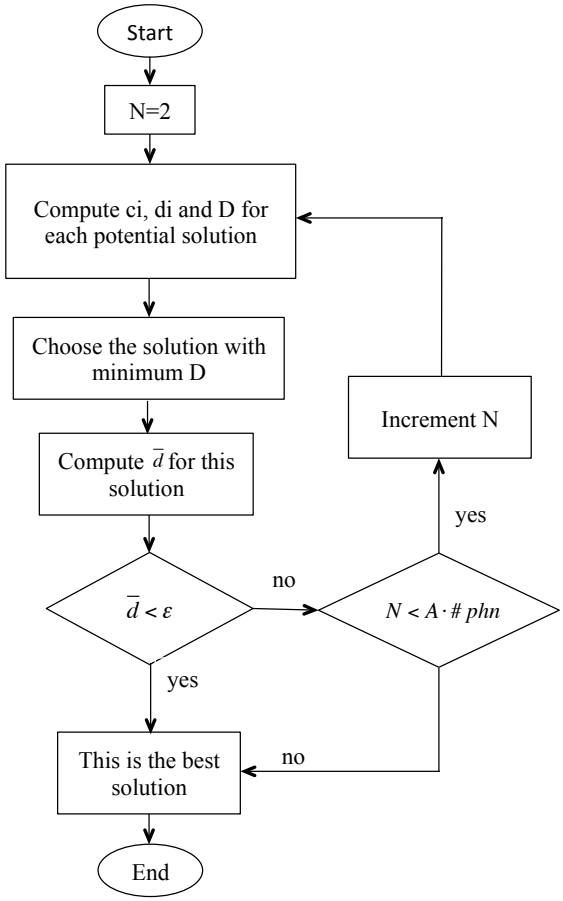


Figure 13: The acoustic segmentation algorithm

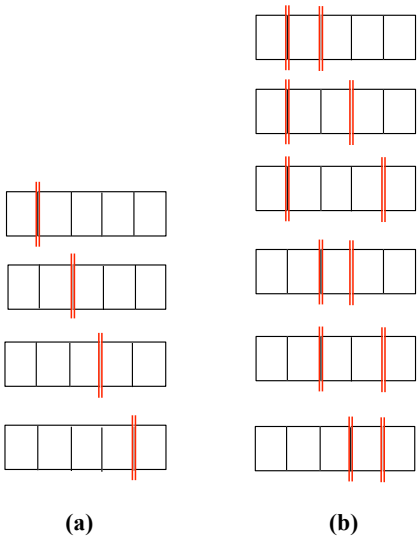


Figure 14: Two- and three- segmentation solutions for a signal with 5 frames

Figure 14 shows the segmentation problem for two- and three-segment solutions. In this figure, the signal consists of 5 frames. So there are 4 potential solutions for two-segment problem and 6 for three-segment problem.

The algorithm is applied to the whole database. So, after segmentation, what we gain is a bundle of segments with various statistical properties.

4.4 Distortion Matrix

As described in the previous section, the algorithm works based on level building from two segments and upward. Since there are a great amount of possible solutions in each level, computing the distortion for all segments in all levels would make the process quite slow and redundant. Hence, the algorithm is implemented such that in order to compute the best segmentation for upper levels, it exploits what it has already calculated in the solution for lower levels. For each utterance, the algorithm sets up a squared distortion matrix of dimension $N_t \times N_t$ (N_t is the number of frames in that utterance) used to record all possible solutions (segment combinations) in order to prevent re-calculation in next levels. This efficient technique is called *dynamic programming*.

4.5 Summary

In this chapter, the posterior vectors generated from the acoustic merging module were used as input to a segmentation engine. First the idea behind the segmentation was described. Then the acoustic segmentation algorithm of speech utterances was included. The result is segments with minimum acoustic internal variations and variable lengths, expanding the whole speech space and represented by their centroid vectors.

5 Data Clustering

In this chapter with the segment representatives (centroids) at hand, the data clustering procedure will be explained.

Data clustering is one of the most popular techniques in pattern recognition problems. The clustering can generally be achieved via two major approaches known as *supervised* and *unsupervised* learning techniques. In supervised learning, the annotated version of training data is available. However, in real applications, such as in conversational speech recognition, it is both expensive and impossible to provide labels such as transcription of the data set in phoneme level. Such problems are regarded as unsupervised learning. In unsupervised learning technique, the only available data is the unlabelled input vectors to be clustered, and the labels will be assigned after the clustering is performed. There is a wide range of techniques employed in unsupervised problems. One of the most common approaches is k-means. This is a clustering algorithm, aiming to partition the data set into k clusters in such a way that the segments possessing similar characteristics are clustered together. One of the major applications of this approach is in data classification tasks. It is also valuable in data compression applications, as the number of candidates (k) is much less than the size of dataset [3].

In this chapter, all principles regarding the k-means algorithm are provided in (5.1) and the re-labeling strategy of the database is described in (5.2). In (5.3), a brief summary of the chapter is presented.

5.1 K-means Algorithm

As mentioned above, the goal is to cluster the data points into k clusters, define k candidates and finally assign the cluster label to each data point. This should be performed such that the distortion measure is minimized. However, it is not an easy task to jointly minimize the distortion with respect to both cluster candidates and cluster labels. So, we need to adapt a two-stage iterative strategy:

- Keep the candidates fixed and assign cluster labels to data points, with respect to the distortion measure minimization.

- Keep the membership assignment fixed and update the candidates.

By this strategy, the distortion minimization is guaranteed [20].

There are several variants of k-means algorithm based on how the candidate assignment and the distortion measure are defined. In case of candidate initialisation, one could randomly assign k data points from the data set as initial candidates. Another alternative is to set the k farthest points as the initial candidates. However, in the current work, since the k is undefined, none of the above mentioned can be applied. Here we have adopted another strategy, described in the following subsection. In case of distortion measure, one could choose the absolute differences between the data points and the candidates (L1 distance). Another alternative, which is employed in this work, is the squared Euclidean distance.
In the following sub-section, the k-means clustering for two-dimensional data set is described. This will provide visual insight into the project work. The corresponding MATLAB script is enclosed in Appendix C.

5.1.1 How The Algorithm Works

Figure 15 demonstrates the scatterplot of a two-dimensional data set while applying k-means. In this figure, the existence of 4 clusters is clear from the way the data points are distributed in space. Now let's see how the algorithm finds the clusters. The process is made up of the following steps:

Initialisation: The algorithm starts by computing the centroid vector corresponding to the whole data set (Figure 15a). This centroid vector is slightly biased (by a random vector) to give another candidate vector. Now we have two candidates; one is the original centroid, and the other is the biased version of the original centroid (Figure 15b).

Recursion: Now, each data point is assigned to its closest centroid and then, for each cluster i, the local distortion error defined as sum of squared Euclidean distances between data point members and candidate vectors is computed:

$$L_i = \sum_{i=1}^{n} |x_i - c_i|^2 \qquad (5)$$

where x_i are the data points belonging to cluster i, c_i is the candidate vectors and n is the total number of data points in that cluster. For each cluster, the candidates are then updated to be the centroid vector of data points belonging to that cluster (Figure 15c).
After that, it is the time to compute the global distortion, defined as

$$G_k^i = \sum_{i=1}^{m} L_i \tag{6}$$

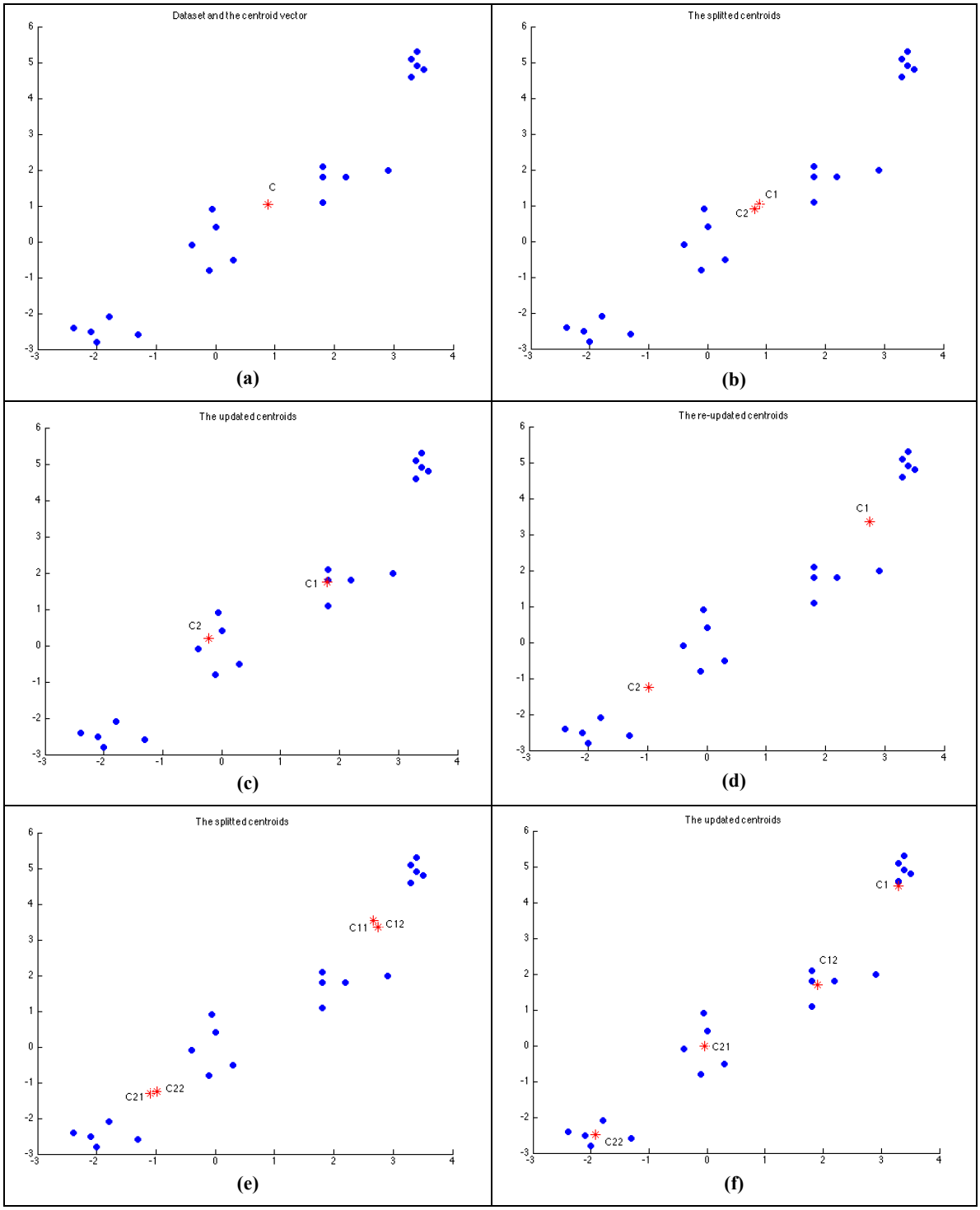
Where m is the current number of clusters. At next step, the membership assignments are re-updated until either no data point changes its cluster membership, or the global distortion improvement defined as

$$\left| \frac{G_{k-1}^i - G_k^i}{G_{k-1}^i} \right| < \theta_1 \tag{7}$$

falls below a certain threshold. In Eq. (6) and (7), k is the stabilization iteration index and i is the splitting iteration index. In case of two-dimensional data provided in this example, this step is iterated two times to converge. At this point, we have two clusters and two stabilized centroid vectors (Figure 15d).

Termination: The question here is when should the algorithm stop generating new clusters? Here again the global distortion error is employed as the stop criterion. The algorithm would stop when the global distortion does not improve much compared to the minimum global distortion of the previous iteration.

$$\left| \frac{G_k^i - \min_l G_l^{i-1}}{\min_l G_l^{i-1}} \right| < \theta_2 \tag{8}$$


Dataset and the centroid vector
C
(a)
The splitted centroids
C1
C2
(b)
The updated centroids
C1
C2
(c)
The re-updated centroids
C1
C2
(d)
The splitted centroids
C11
C12
C21
C22
(e)
The updated centroids
C1
C12
C21
C22
(f)

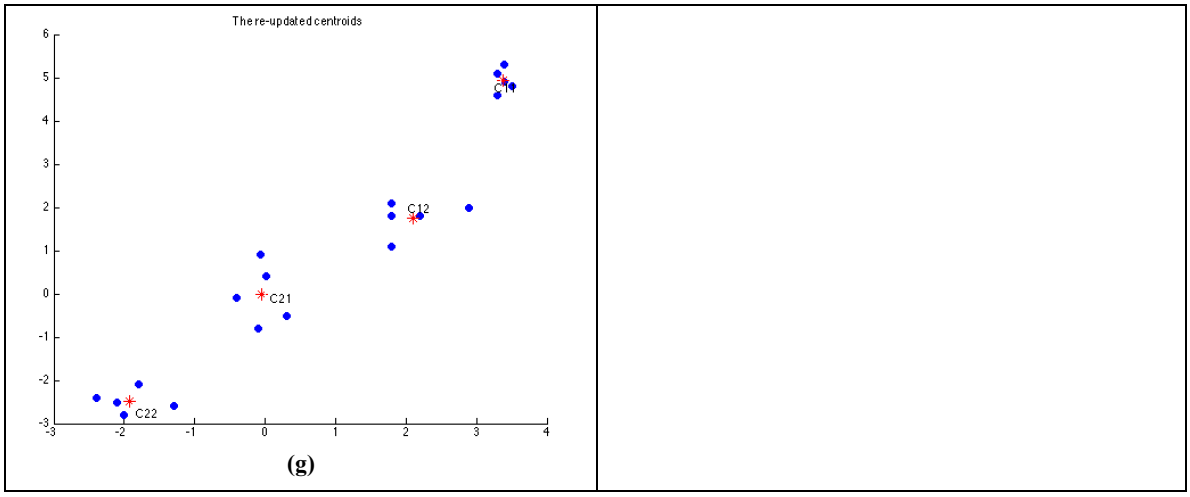


Figure 15: Illustratin of the k-means algorithm in a two-dimensional Euclidean space

5.2 Labelling Assignment Strategy

The outputs from the clustering algorithm, i.e. the cluster labels, are used to annotate the segments of the database in terms of the cluster labels. In this section we will describe the re-labeling principle of the utterance segments by an example.

Figure 16(left) shows the three utterances after being segmented via the acoustic segmentation algorithm. In this figure c_i, c'_j and c''_z are representing the representative vectors of utterance segments and S_i, S'_j and S''_z are corresponding to the primary segment labels.

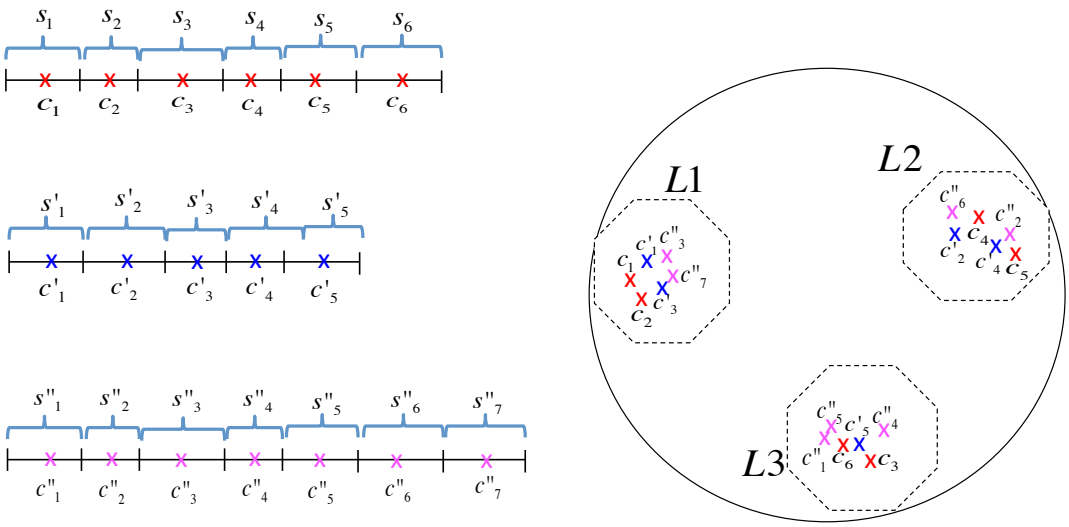


Figure 16: Labelling principle. Database architecture (left) and centroid space (right)

Figure 16(right) shows how the location of these representative vectors in space has made the existence of 3 clusters evident. After clustering, the clusters are tagged by L1, L2 and L3. These tags are then used to label the members belonging to each cluster. These cluster members are representatives of segments in database. Hence, each segment is now re-labeled by cluster labels, as shown in Figure 17. In a similar fashion, the whole database is transcribed by a new set of labels, each of which a cluster label.

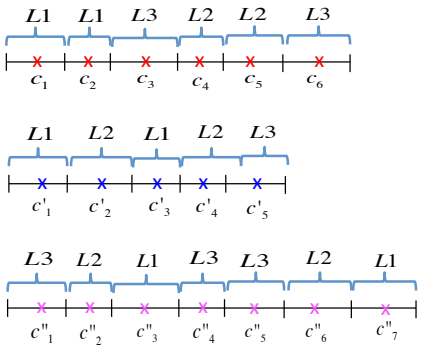


Figure 17: Database after label assignment

5.3 Summary

In this chapter, the principle of clustering the segmental centroids by applying k-means algorithm was described. The objective function was to minimize the distortion error, which was selected to be sum of the squared Euclidean distances between all data points and their nearest centroid vectors. The output from the clustering algorithm constructed a re-labeled version of the segments in terms of the cluster labels.

6 Acoustic Modeling

In speech recognition, if we assume that the recognizer is designed to recognize basic recognition units, and if the acoustic observation sequence corresponding to the whole speech signal is coded as $O = (o_1, o_2, o_3, .., o_t)$, then the primary task of recognizer is to find the most probable sequence of linguistic units U for observation sequence O. In other words, the goal is to find recognition unit U such that the following conditional probability is maximized:

$$\hat{U} = \arg\max_{U} P(U|O) \tag{9}$$

The probability on the right hand side of Eq. (9) is not directly computable. However, it can be converted to a more feasible form according to the Byes' rule:

$$P(U|O) = \frac{P(O|U).P(U)}{P(O)} \tag{10}$$

The observation probability $P(O)$ in the denominator is not depending on the speech unit U, and hene can be skipped. So, the recognition task is simplified as:

$$\hat{U} = \arg\max_{U} P(O|U)P(U) \tag{11}$$

According to Eq. (11), the most probable recognition unit is found by computing the product of two probabilities on the right hand side of equation – known as the *likelihood* and the *prior* probability, respectively – and picking up the recognition unit that maximizes this product. Computing the likelihood and prior probabilities directly from training data is infeasible. However, they can be estimated from parametric statistical models known as *acoustic* and *language* models, respectively.

In this chapter we will present the idea behind acoustic modeling. A common technique in computing the likelihood probabilities and modeling acoustics of speech is Hidden Markov Model (HMM). In this case, each basic unit is represented by a single HMM model. The

parameters of this HMM are estimated from the speech data, and the likelihoods are computed [7].

The chapter is structured as follows: First, the concept of Markov model, considered as the fundamental tool to understand HMMs is introduced. (6.1) Then the chapter presents the theory regarding HMMs (6.2), and their structure is described in (6.3). In (6.4), three well-known HMM problems are described and for each one, the common solution algorithm is introduced. The chapter ends with giving a summary at (6.5).

6.1 Markov Model

A Markov model is essentially a model consisting of a set of states $S = (s_1, s_2, \dots, s_N)$ each generating an observation vector. In this model it is known which state generates which observation vector. Each state in such model depends only on the n previous states in the model. In its simplest form:

$$P(s_i|s_1^{i-1}) = P(s_i|s_{i-1}) \qquad (12)$$

In this case, the model called the first order Markov assumption, and can be interpreted as follows: The probability of being in state i in a particular time depends only on the state of the model at the previous time.

A Markov model is defined by the following parameters:

- $\boldsymbol{\pi_i}$ - The probability to start from state i, where $\sum_{i=1}^{N} \pi_i = 1$
- $\mathbf{a_{ij}}$ – The set of transiting probabilities, i.e. the probability of moving from state i to state j, which satisfies $\sum_{i=1}^{N} a_{ij} = 1$. These are stored in the form of a transition matrix A.
- $\mathbf{b_j(o_t)}$ - The probability of generating a particular observation vector o_t from state j. This probability is limited to the values 1 and 0; if the state has generated the observation it is 1, otherwise it is 0 [21].

A first order Markov model with three states is shown in Figure 18. In this example, a_{12} is equal to one, while a'_{23} and a''_{23} have other values summing up to one. The observation probabilities $b_1(o_1)$, $b_2(o_2)$ and $b_3(o_3)$ are equal to one.

The Markov model is a powerful tool in probabilistic problems. However, this model is not used in speech recognition, as it is not able to model the variable nature of the speech signal. As a solution to this problem, the hidden Markov Model (HMM) is introduced in next section.

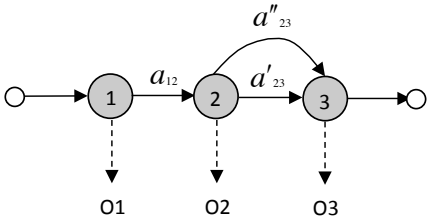


Figure 18: The scheme of a Markov model

6.2 Hidden Markov Models

A Hidden Markov models is a powerful statistical model for modeling a data sequence that is not predictable based on a set of observations. It is largely applied in many tasks such as speech recognition systems, protein/DNA sequence analysis, robot control, and information extraction form text data [23]. This model can be described as a generalization of the markov model with some major differences. Firstly, in case of HMM, its not known which state has emitted which observation. The reason is simply because the states are not observable and not known to the system anymore. Secondly, in HMM, each state has the ability to not only follow the other states, but also loop back to itself. This confirms the fact that HMMs are capable of modeling the various time durations of the speech recognition units. [21] Thirdly, the observation probabilities in HMM are not limited to just 0 and 1, but can take on any value form 0 to 1, as to be described in the following section.

6.3 HMM Structure

Two typical structures of HMM are depicted in Figure 19. In this figure, the emitting states are shown by circles, the non-emitting states by nodes, the transition probabilities by arcs (edges) between states and loops to the states, and the emission probabilities by arcs drawn out of states.

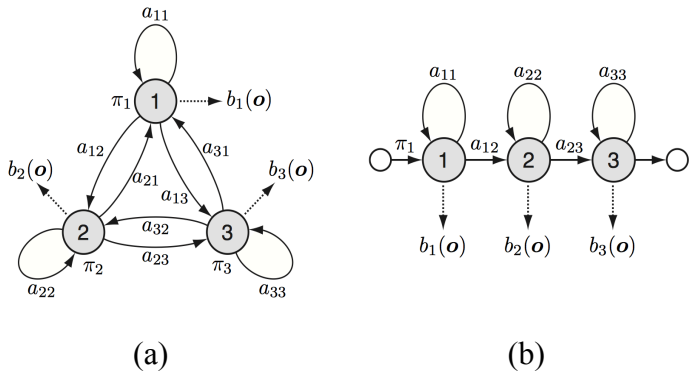


Figure 19: Typical HMM structures: (a) Ergodic (b) Left-to-right

Figure 19(a) shows a 3-state ergodic HMM, in which each state has the potential to start the process, and change to other states at subsequent time steps. In Figure 19(b), an example of a three state left-to-right HMM model is shown. In this model, state 1 is the starting node. At each time point, the state index is either increased by one unit or remained unchanged, and then observation vectors according to the output probability distribution of the current state are generated. This model is commonly used in speech recognition tasks, firstly because the random selection of the initial state is not the case in speech recognition, secondly because it satisfies the natural progression of speech signal in time domain, and thirdly because it can successfully model the time variation nature of the speech signal by staying at the same state for a period of time.

A single HMM model is defined by the following parameters:

$\boldsymbol{\pi_i}$ - The probability to start from state i, where $\sum_{i=1}^{N} \pi_i = 1$

$\mathbf{a_{ij}}$ - The probability of transiting from state i to state j, which satisfies $\sum_{i=1}^{N} a_{ij} = 1$. These are represented as components of transition matrix A.

$\mathbf{b_j(o_t)}$ - The probability of generating observation vector o_t from state j, where $\sum_{t=1}^{T} b_j(o_t) = 1$ for $i = (1,2,\ldots,N)$. The set of these observation likelihoods is the emission matrix B.

The parameters that completely define the model are M = {π, A, B}.

The number of states is also considered as a user-defined parameter and is selected according to the task. The HMM models used in this work are one-state models with one emitting state at middle, as shown in Figure 20.

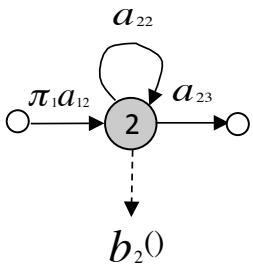


Figure 20: one-state left to right structure for basic units in this work

The choice of output distribution function in order to estimate emission probabilities $b_j(o_t)$ is crucial in recognition performance. In case of continuous speech, a common choice is single component, multivariate Gaussian distribution, that is:

$$b_j(o_t) = P(o_t|s_t = j) = \mathcal{N}(o_t; \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} e^{-\frac{1}{2}(o_t-\mu_j)^T \Sigma_j^{-1} (o_t-\mu_j)} \quad (13)$$

where μ_j is mean and Σ_j is the covariance matrix defined as:

$$\hat{\mu}_j = \frac{1}{T}\sum_{t=1}^{T} o_t \quad (14)$$

$$\hat{\Sigma}_j = \frac{1}{T}\sum_{t=1}^{T} (o_t - \mu_j)(o_t - \mu_j)^T \quad (15)$$

A more flexible alternative for density estimation of observation probabilities is the mixture Gaussian distribution. A Gaussian mixture model (GMM) is a weighted sum of M single Gaussians,

$$b_j(o_t) = \mathrm{P}(o_t|s_t = j) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm}) \quad (16)$$

where M is the number of mixture components and c_{jm} is the mixture weight parameter which satisfies:

$$\sum_{m=1}^{M} c_{jm} = 1 \tag{17}$$

6.4 Classic HMM Problems

In each speech recognition task there are three problems that need to be solved. In the following subsections, the problems and their corresponding efficient solutions are described [18].

6.4.1 Probability Evaluation

In this problem, the HMM model M and also the observation sequence $O = (o_1, o_2, \dots, o_T)$ are known, and the goal is to find the probability of the observation sequence being generated. This could be found by summing up all the probabilities of all the state sequences that generate the observation sequence:

$$P(O|M) = \sum_{S=s_1}^{s_T} P(S|M)P(O|S,M) \tag{18}$$

The probability in Eq. (18) consists of two parts, the first component is the state-sequence probability and for a specific state sequence $S = (s_1, s_2, \dots, s_T)$ could be rewritten as:

$$P(S|M) = P(s_1|M)\prod_{t=2}^{T} P(s_t|s_{t-1}, M) = \pi_{s_1} a_{s_1 s_2} \dots a_{s_{T-1} s_T} \tag{19}$$

The second component in Eq. (18) could be rewritten as:

$$\begin{aligned} P(O|S,M) &= P(O_1^T|S_1^T, M) = \prod_{t=1}^{T} P(o_t|s_t, M) \\ &= b_{s_1}(o_1) b_{s_2}(o_2) \dots b_{s_T}(o_T) \end{aligned} \tag{20}$$

Substituting Eq. (19) and (20) into (18) yields:

$$\begin{aligned} P(O|M) &= \sum_{S=s_1}^{s_T} P(S|M)P(O|S,M) \\ &= \sum_{S=s_1}^{s_T} \pi_{s_1} b_{s_1}(o_1) a_{s_1 s_2} b_{s_2}(o_2) \dots a_{s_{T-1} s_T} b_{s_T}(o_T) \end{aligned} \tag{21}$$

Direct evaluation of Eq. (21) is not feasible, as it involves high computational complexity. Instead, an efficient algorithm called the *Forward algorithm* is employed. This is an iterative algorithm and operates based on dynamic programming technique described in (4.4).

In this algorithm, the joint probability of being in state j at time t is denoted as the Forward parameter $\alpha_t(j)$ and formulated as:

$$\alpha_t(\mathrm{i}) = \mathrm{P}(\mathrm{O}_1^t, \mathrm{s}_t = \mathrm{i}|\mathrm{M}) \tag{22}$$

The Forward algorithm includes the following steps:

1. Initialisation: A forward parameter is computed at time 1 for all states by:

$$\alpha_1(\mathrm{i}) = \pi_\mathrm{i} \mathrm{b}_\mathrm{i}(\mathrm{o}_1) \tag{23}$$

where π_i is the initialisation probability at state i.

2. Induction: For all states $\mathrm{j} = (1, 2, \ldots, \mathrm{N})$ at all other time points $\mathrm{t} = (2,3, \ldots, \mathrm{T})$, this probability is found by:

$$\alpha_t(\mathrm{j}) = \left[\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{a}_{\mathrm{ij}} \alpha_{\mathrm{t}-1}(\mathrm{i})\right] \mathrm{b}_\mathrm{j}(\mathrm{o}_\mathrm{t}) \tag{24}$$

3. Termination: The algorithm ends by summing up all the forward probabilities at final time T:

$$P(\mathrm{O}|M) = \sum_{i=1}^{N} \alpha_T(i) \tag{25}$$

This probability evaluation is mostly useful to analyse the performance of the system after training (the term 'training' to be explained later).

6.4.2 Optimal State Sequence

The second interesting problem is about to find out which sequence of states has most probably generated the observation sequence. This problem is known as decoding, and

involves finding a solution for Eq. (11). Here, again, the HMM model parameters and the observation sequence are known to the system. The most common algorithm for performing decoding is the Viterbi algorithm. This algorithm exploits first-order Markov property by which at each state only the most probable path is kept. Here, the maximum probability of being in state j at time t is denoted as the Viterbi parameter $V_t(j)$ and formulated as:

$$\mathrm{V_t(j)} = \max_{\mathrm{S_1^{t-1}}}(\mathrm{P(O_1^t, S_1^{t-1}, s_t = j)|M}) \tag{26}$$

This implies keeping the most probable path and discarding the rest until the final time. The algorithm also records the indexes of the most probable state at each step in matrix B. This is again a recursive algorithm with the following steps:

1. Initialisation: The Viterbi parameter and the index recorder for all states $i = (1,2,\dots,N)$ at time 1 are initialised as:

$$\mathrm{V_1(i)} = \pi_\mathrm{i} b_\mathrm{i}(o_1) \tag{27}$$

$$\mathrm{B_1}(i) = 0 \tag{28}$$

2. Induction: For all states $\mathrm{j} = (1, 2, \dots, \mathrm{N})$ at all other time points $\mathrm{t} = (2,3,\dots,\mathrm{T})$, the Viterbi value and the index recorder are:

$$\mathrm{V_t(j)} = \max_i\left[\mathrm{V_{t-1}(i)a_{ij}}\right] \mathrm{b_j}(o_\mathrm{t}) \tag{29}$$

$$B_t(j) = arg\max_i\left[V_{t-1}(i)a_{ij}\right] \tag{30}$$

3. Termination: The most probable final path is found by computing the Viterbi approximation for all states at final time T, and selecting the highest probability value:

$$\text{The best score} = \max_i\left[V_T(i)\right] \tag{31}$$

$$S_T^* = \operatorname{argmax}_i\left[B_T(i)\right] \tag{32}$$

4. Backtracking: In this final step, a backtracking is performed to find the most probable

path:

$$s_t^* = B_{t+1}(s_{t+1}^*) \qquad t = T-1, T-2, .., 1 \tag{33}$$

The best sequence will then be:

$$S^* = (s_1^*, s_2^*, s_3^*, .., s_T^*) \tag{34}$$

In order to prevent underflow in decoding operation, the calculations are typically performed in log domain. Eq. (29) in log domain would be:

$$V_t(j) = \max_i \left[V_{t-1}(i) + \log a_{ij}\right] + \log\left(b_j(o_t)\right) \tag{35}$$

As mentioned above, the Viterbi algorithm described here provides a solution to Eq. (11), and hence performs an exact search among all possible state sequences in the model. However, an exact search is not efficient in large vocabulary tasks and leads to extremely high computational time – and memory – requirement. So the main problem in recognition would be to reduce the search space. As a solution to this problem, one can apply beam search strategy by which paths with low chance to succeed – these are the paths with probability less than the best path within a defined factor $\propto$ – are cut away and only the most promising paths are considered to be active. This is regarded as a sub-optimal solution since the optimal hypothesis may be incorrectly pruned. However, by choosing an appropriate value for pruning beam-width, it is high probable to find the best path.

6.4.3 Parameter Estimation

In this task the goal is to estimate the HMM model parameters, such that the probability of generating a sequence of observation vectors $O = \{o_1, o_2, .., o_T\}$ is maximized:

$$\widehat{M} = arg \max_M P(O|M) \tag{36}$$

The model parameters are those introduced in section (6.3). In order to estimate these parameters, an iterative algorithm referred to as *Baum-Welch* (*forward-backward*) algorithm is used. The forward probability was described in (6.4.1). Before describing the Baum-Welch

algorithm, it is necessary to introduce several variables. The first variable is the *backward* parameter $\beta_t(j)$ defined as

$$\beta_t(j) = P\left(O_{t+1}^T | s_t = j, M\right) \tag{37}$$

This is the probability of partial observation sequence O_{t+1}^T, given that the system is in state j in time t. This is again a recursive algorithm, comprising the following steps:

1. Initialisation: The initial condition is given by:

$$\beta_T(i) = \frac{1}{N} \tag{38}$$

2. Induction: for all other time points $t = (T-1, \ldots, 1)$, and all states $i = (1, \ldots, N)$ the Backward probability is found by

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\beta_{t+1}(j)b_j(o_{t+1}) \tag{39}$$

The second useful variable is $\gamma_t(i,j)$,

$$\gamma_t(i,j) = P(s_{t-1} = i, s_t = j | O_1^T, M)$$

$$= \frac{P(s_{t-1} = i, s_t = j, O_1^T | M)}{P(O_1^T | M)}$$

$$= \frac{\alpha_{t-1}(i) a_{ij} b_j(O_t) \beta_t(j)}{\sum_{k=1}^{N} \alpha_T(k)} \tag{40}$$

This is the transition probability form state i to state j at time t, given the observation sequence and the model parameters.

The algorithm aims to maximize the following auxiliary function:

$$\varphi\left(M,\widehat{M}\right) = \sum_{all\ S} \frac{P(O,S|M)}{P(O|M)} \log P(O,S|M^*) \tag{41}$$

Eq. (41) is a separable equation, and can be rewritten as

$$\varphi\left(M,\widehat{M}\right) = \varphi_a(M,\hat{a}) + \varphi_b\left(M,\hat{b}\right) \tag{42}$$

with,

$$\varphi_a(M,\hat{a}) = \sum_i \sum_j \sum_t P(s_{t-1}=i, s_t=j|\, O,M) \log \hat{a}_{ij} \tag{43}$$

$$\varphi_b\left(M,\hat{b}\right) = \sum_j \sum_t P(s_t=j|\, O,M) \log \hat{b}_j(O_t) \tag{44}$$

Eq. (43) and (44) are both of the form,

$$F(x) = \sum_i y_i \log x_i \tag{45}$$

which is maximised when,

$$x_i = \frac{y_i}{\sum_i y_i} \tag{46}$$

Thus,

$$\hat{a}_{ij} = \frac{\frac{1}{P(O|M)}\sum_{t=1}^{T} P(O,s_{t-1}=i,s_t=j|M)}{\frac{1}{P(O|M)}\sum_{t=1}^{T} P(O,s_{t-1}=i|M)} = \frac{\sum_{t=1}^{T}\gamma_t(i,j)}{\sum_{t=1}^{T}\sum_{k=1}^{N}\gamma_t(i,k)} \tag{47}$$

$$\hat{b}_j(k) = \frac{\frac{1}{P(O|M)}\sum_{t=1}^{T} P(O,s_t=j|M)\delta(O_t,o_k)}{\frac{1}{P(O|M)}\sum_{t=1}^{T} P(O,s_t=j|M)} = \frac{\sum_{t\epsilon O_t=o_k}\sum_i \gamma_t(i,j)}{\sum_{t=1}^{T}\sum_i \gamma_t(i,j)} \tag{48}$$

Now with forward and backward probabilities, the auxiliary function $\varphi\left(M,\widehat{M}\right)$ and the parameter re-estimators $\hat{a}_{ij}$ and $\hat{b}_j(k)$ in place, it is possible to describe the

Baum-Welch algorithm. This is again a recursive algorithm involving the following steps:

- Step 1: Choose an initial estimate of the model parameters.
- Step 2: Compute the auxiliary function $\varphi(\mathrm{M}, \widehat{\mathrm{M}})$ based on M.
- Step 3: Calculate $\widehat{\mathrm{M}}$ according to the probability re-estimators in Eq. (47) and (48).
- Step 4: Iterate from step 2 until convergence.

6.5 Summary

In this chapter, the theory of acoustic modeling – the module representing the relationship between the speech signal and the basic linguistic units constructing the speech – was described. Then the theory regarding HMM as one of the most frequently used acoustic models was described. Three well-known problems in HMM and the corresponding solution algorithm for each problem were also provided.

7 The Hidden Markov Model Toolkit

In the previous chapter, the basic idea of HMMs and their use in speech recognition tasks was outlined. The current chapter deals with the theory regarding the *Hidden Markov Model toolkit* (HTK). This is a toolkit consisting of a set of library modules and programming tools for building and manipulation of HMMs. The HTK is primary intended to be used in speech recognition tasks, however it could be used as a general-purpose toolkit to build HMMs that model any time series [22].

HTK consists of a set of library modules and tools available in C source form. HTK tools can be run from a command line. However, in large applications with many files, its more common to provide all tools in a script and so execute them by running the script. Figure 21 shows an example of running a HTK tool. The options prefixed by a minus sign, such as -f and -m in this example are optional arguments, and the options consisting of a capital letter, such as -A, -D, -C, -M in this example are common across all HTK tools.

The procedure of building a speech recognizer can be divided into five main phases; creating a prototype HMM model (7.1), initialisation (7.2), re-estimation (7.3), recognition (7.4) and analysis (7.5). In (7.6) two other HTK tools employed during the work are described. Finally the chapter is summarized in (7.7). All the information presented in this chapter are adopted from [22].

```
HCompV -A -D -T 1 -C config -f 0.01 -m -S train.scp -M hmm0 proto
```

Figure 21: Invoking a HTK tool

7.1 Creating a Prototype HMM Definition

The first step in designing acoustic models is to write a prototype HMM model – using a text editor – and storing it as a text file. The prototype HMM definition file includes the following:

- The model name
- Type of observation vector
- Number of internal states
- For each emitting state

  – mean and variance components
  – mixture component weights (in case of Gaussian mixture models)

- Transition matrix

The prototype definition is just a primary representation of the HMM model with arbitrary parameter values. These parameters are to be estimated later during both initialisation and training processes. The means and the variance parameters are usually set to zero and one, respectively. The transition matrix is typically constructed such that transitions out of the emitting states (here from state 2 to itself and to state 3) are equally probable. This is a square matrix, with number of rows and columns equal to number of internal states, with each row summing up to 1 except for the final row which sums to 0 since no transitions are allowed to exit from the final state.
The model name should consist of alphabetic/alphanumeric characters and must not be only numbers. In this document, HMM models are named by Li, where i is an integer.

7.2 Initialisation

The second step of building a speech recogniser deals with initialising the HMM models. This can be done using either of the two available programming tools, HCompV and HInit. They are described in the following subsections. An invocation example of each tool together with a description of the required command line options is provided in next chapter.

7.2.1 Flat Start Initialisation using HCompV

In this approach, all the model parameters corresponding to all emitting states are initialized by the global mean and variances of all training data. Applying HCompV, all HMM models are initialised identically. The procedure is depicted in Figure 22.

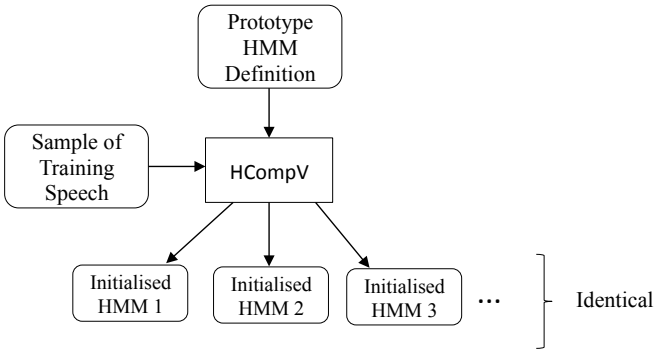


Figure 22: Flat start initialisation of HMMs using HCompV

7.2.2 Isolated Model Initialisation Using HInit

HInit can be used when the transcription is provided such that the time-boundary information for each unit is available. In this case the HMM models are trained individually, as shown in Figure 23. HInit is an iterative procedure with steps shown in Figure 24. At first iteration for each HMM to be initialised, all occurrences of the model units corresponding to that HMM are cut out from the transcription data, and the corresponding feature vectors are distributed equally among the model states, through the uniform segmentation approach.

On the next cycles, the data vectors are assigned to the most probable state found by Viterbi algorithm. This process is called *Viterbi segmentation*. The mean and variance parameters are again estimated by averaging all data vectors associated with each state. The transition probabilities are also estimated by counting the number of times each state was occupied. This process is performed iteratively until either a defined number of iterations is achieved, or the parameter values do not change anymore.

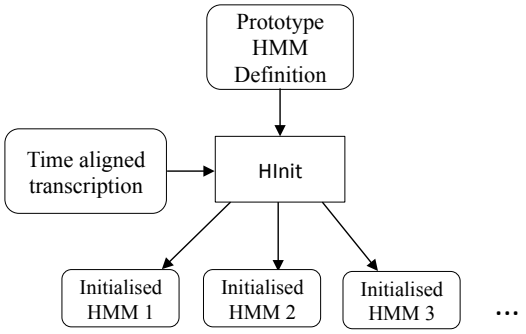


Figure 23: Isolated initialisaiton using HInit

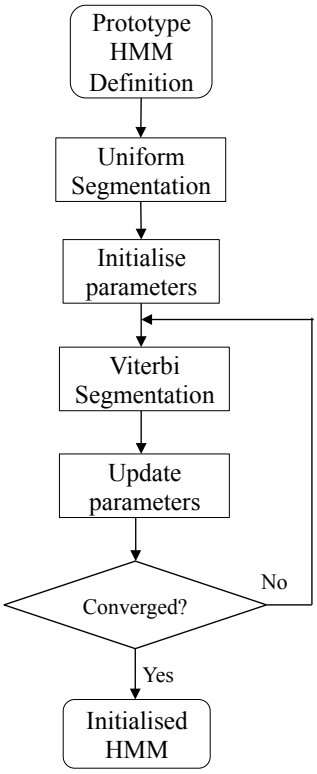


Figure 24: Schematic diagram of HInit algorithm

7.3 Re-estimation

After that the HMM models are initialised, they are trained by either of the tools HERest or HRest, in order to get the model parameters retrained.

7.3.1 Embedded Training Using HERest

This tool is applied in conjunction with the initialisation tool HCompV described above. In this approach, all initialised HMM models are updated at the same time, using all training data. The whole process is performed in one iteration employing the embedded version of Baum Welch re-estimation approach.

In order to get accurate acoustic models, a large amount of training data is needed. However, processing a large amount of training data in a single iteration would be quite time-consuming. In order to improve the computational speed, HERest reduces the size of search speech by restricting the computation of the forward probabilities to those for which

$\alpha_j(t)\beta_j(t)$ is within a fixed distance from the total likelihood $P(O|M)$. This process is called *pruning*. Pruning can be further controlled by the user by activating the command line option -t to the backward probabilities. For example:

```
HERest -A -D -T 1 -C config -I labels_train.mlf -t 150.0 100.0 500.0 -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 labellist
```

In this example, the training starts by a beam width equal to 150. If a pruning error occurs for an utterance, the beam is changed to 100 and that utterance is reprocessed. This would continue until no error is observed, or the maximum beam limit 500 is reached.

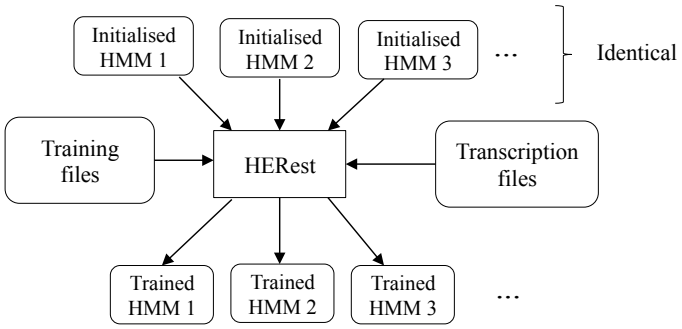


Figure 25: Training process using HERest

7.3.2 Isolated Training Using HRest

This tool is employed in conjunction with the isolated model initialisation tool HInit. As shown in Figure 26, It takes the same data as HInit, but instead of applying first uniform segmentation and then iteratively Viterbi alignment, it iteratively employs the Baum-welch approach. The procedure is illustrated in Figure 27.

It is important to note that although both HRest and HERest employ Baum Welch algorithm, the embedded training program HERest does not consider the label boundary information, as for each training data, the whole file is processed at once. This is not the case for HRest, in which as described earlier, the training data is cut out into boundaries corresponding to transcriptions.

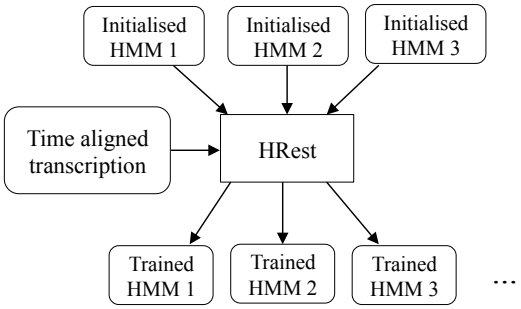


Figure 26: Training by HRest

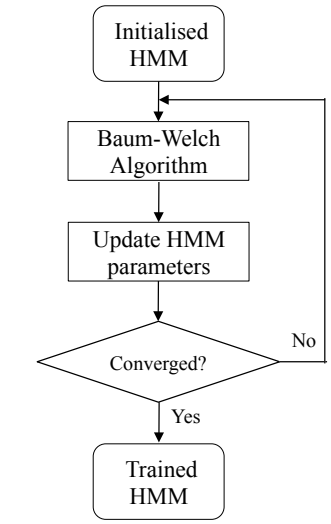


Figure 27: Schematic diagram of HRest algorithm

7.4 Decoding

After that the models are trained, they are ready to perform decoding. As mentioned in introduction chapter, because of time constraints, in this work the decoding is just investigated in basic unit level. Therefore, the higher-level recognition is not described.

Decoding involves employing the trained HMMs to transcribe a collection of unknown utterances. In order to perform HMM-level recognition, the following data are required:

- HMM-level network
- Dictionary
- The HMM models

Decoding is performed by the HTK library module HRec adopting token passing paradigm to find the best path. To drive HRec from command-line, the tool HVite can be employed.

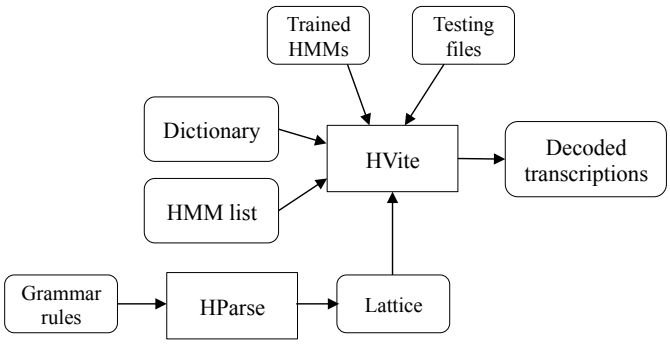


Figure 28: Decoding procedure using HVite

7.4.1 The Token Passing Algorithm

A token represents a partial path through the network extending from the time point 0 to time point t. In this algorithm, the model is initialised by placing a token at all states in time 0. Each token then passes around the transition network, and at each emitting state its value is incremented by the emission and transition probabilities. Each network node is allowed to keep at most N tokens. Therefore, at the end of each time step, only the N best tokens in any node are kept.

HVite has many parameters with direct impact to the recognition performance. For instance, in order to reduce computational time, a pruning beam-width is considered to be controlled by the user. This value shouldn't be set too small as this would result in pruning the most probable path, and so leading to recognition error.

7.4.2 Forced Alignment

Forced alignment can be seen as a pre-step to the isolated training process. In this approach, the boundary locations of the segments in transcription files are aligned with the speech data, and hence a new set of time boundaries for the segments are specified. This is again provided by the recognition tool HVite by activating the option -a. The procedure is shown in Figure 29.

In forced alignment, by employing the dictionary and the transcriptions a new network for each utterance is constructed. This network includes all possible pronunciations of each recognition unit in parallel. Then, employing the Viterbi algorithm, the best matching path

through the network is found and a lattice is generated. This will be converted to a transcription file containing the boundary information of recognition units. After forced alignment, the results are stored in a text file with an aligned version of time indices for each segment.

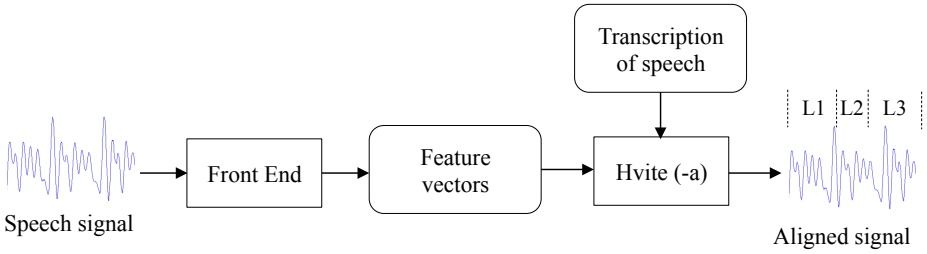


Figure 29: Forced alignment procedure

7.5 Evaluation

In order to analyse the performance of the decoding module, HTK employs an evaluation tool called HResults, which performs a comparison between the reference transcriptions and the recognition results, and counts all occurrences of deletion errors (D), substitutions errors (S), Insertions (I), and computes the following:

$$Percent\ Correct = \frac{N - D - S}{N} \times 100\% \tag{49}$$

$$Percent\ Accuracy = \frac{N - D - S - I}{N} \times 100\% \tag{50}$$

The result will be reported in a text file. The procedure is schematized in Figure 30

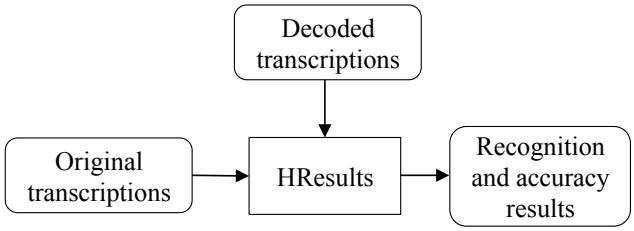


Figure 30: Evaluation using HResults

7.6 Other HTK tools

In this work, two other HTK tools were also used. In the following they are described to the extent they were employed during the experiments.

7.6.1 HLEd

In order to collect all label files into one single file, one can use the label editor tool HLEd. The resulting file is known as *master label file* (MLF). This file starts by a line containing the string #!MLF!# which identifies the file as an MLF file. Each label file corresponding to the transcriptions of a speech file appears as a string (of the form "*/filename.lab") followed by labels in individual lines, and terminated by a period on a line of its own.

7.6.2 HHEd

This tool works in a similar way to HLEd. It can be used to collect all HMM definition files into one single file. The resulting file is called *master macro file* (MMF).

7.7 Summary

In this chapter, the HTK toolkit, known as a powerful tool in building and training HMMs was introduced. Then the operation of initializing, training, testing and analysis of models in both isolated and embedded form was described.

8 Experiments, Results and Discussions

Up to this point, the previous chapters involved all necessary theoretical background in order to perform the experiments. In this chapter the experiments associated with this project are reported and discussed. The chapter starts by a brief introduction of the TIMIT database employed during the experiments (8.1). Then the front-end analysis and detection parameters adopted in this work are represented (8.2). In (8.3), the experiment regarding the acoustic merging is discussed. Section (8.4) deals with the acoustic segmentation implemented in this project and (8.5) discusses the k-means algorithm in more detail. In (8.6) two experiments regarding recognition and forced alignment are described. Finally, (8.7) includes the summary of the chapter.

8.1 The TIMIT Database

The database employed in this work is the Texas Instrumental Massachusetts Institute of Technology (TIMIT) corpus of read speech. It is a high-quality, multi-speaker and multi-accent corpus, including phonemic transcription. In this project, these transcriptions were used for evaluating the results obtained after acoustic segmentation process.

The data is telephone-quality speech, recorded by a Sennheiser HMD 414 headset-mounted microphone, sampled at 16kHz and digitized with 16-bit sample resolution. It consists of totally 6300 sentences. In total there are 630 speakers, each reading 10 sentences, from 8 major dialect regions of the United States.

The signals have been partitioned into training and testing databases (3696 signals for training and 1344 signals for testing) existing in train and test subdirectories respectively. Each utterance is specified by a code, called TIMIT code, representing the speaker sex, the speaker ID and the sentence ID. For instance, *fcjf0_sa1* means that the speaker is a female, with ID number cjf0, and the sentence to be uttered is sa1. [5]

8.2 Front-End Analysis and Detection

The principle of front-end analysis and the acoustic detection were described thoroughly in sections (2.3) and (3.3). Recalling Figure 5, both steps of front-end analysis, namely feature

extraction and STC generation and also acoustic detection were performed in the previous work in specialization project. Table 2 shows the basic required settings employed in this work. Table shows that the primary feature vectors are selected to be mel-bank energy (MBE) extracted from the raw speech signals, using 25 ms Hamming window and 10 ms window shift. There is no pre-emphasis filtering used in the system. The generated MBE feature vectors are then used as input to STC engine to generate STC feature vectors. These are then applied as input to left and right context neural networks with two nodes at the output layer and one hidden layer of 500 nodes. The last two rows in table specify the input feature for networks and the temporal DCT order used in acoustic detection module.

Table 2: The front-end configurations

Parameter	Value
SOURCEKIND	WAVEFORM
SOURCEFORMAT	WAV
TARGETFORMAT	HTK
TARGETKIND	FBANK
NUMCHANS	23
USEPOWER	T
USEHAMMING	T
PREEMCOEF	0
TARGETRATE	100000
WINDOWSIZE	250000
NETWORK INPUT	STC
TEMPORAL DCT	10

8.3 Acoustic Merging

We started the experiments by acoustic merging. As described in section (3.4), this step was about taking all posterior probabilities corresponding to each frame, generated at the output of the bank of detectors, and stacking them in a supervector of dimension 44. The dimension of the supervectors was reduced to 22, in order to reduce the redundant information.

These supervectors construct the basic elements in recognition process. Therefore, their characteristics should be studied to see whether they are qualified to be used as such. In this section we will visually inspect the posterior probabilities and discuss their characteristics.

In order to study the characteristics of posterior vectors we make use of a representation technique, referred to as *posteriorgram*. This is a time- vs.- posterior probability representation by which one can infer how the temporal content of the posterior probabilities is changed over time and also across the detectors.

Figure 31 shows the posteriorgram representation of the utterance “It suffers from a lack of unity of purpose and respect for heroic leadership.” with TIMIT code fadg0_si649.

Here, we have also plotted the binary phonemic labels in red, in which 0 and 1 identify the existence and absence of AF features in speech frames, respectively.

By comparing the results with the manual labelling shown in red, one can conclude that the overall result is promising: There is a relatively high correspondence between the acoustic-phonetic events (posterior probabilities) and the binary labelling. This is specially the case for Silence, Vocalic, Retroflex, Labial, Nasal, Anterior, Vowel and Fricative detectors.

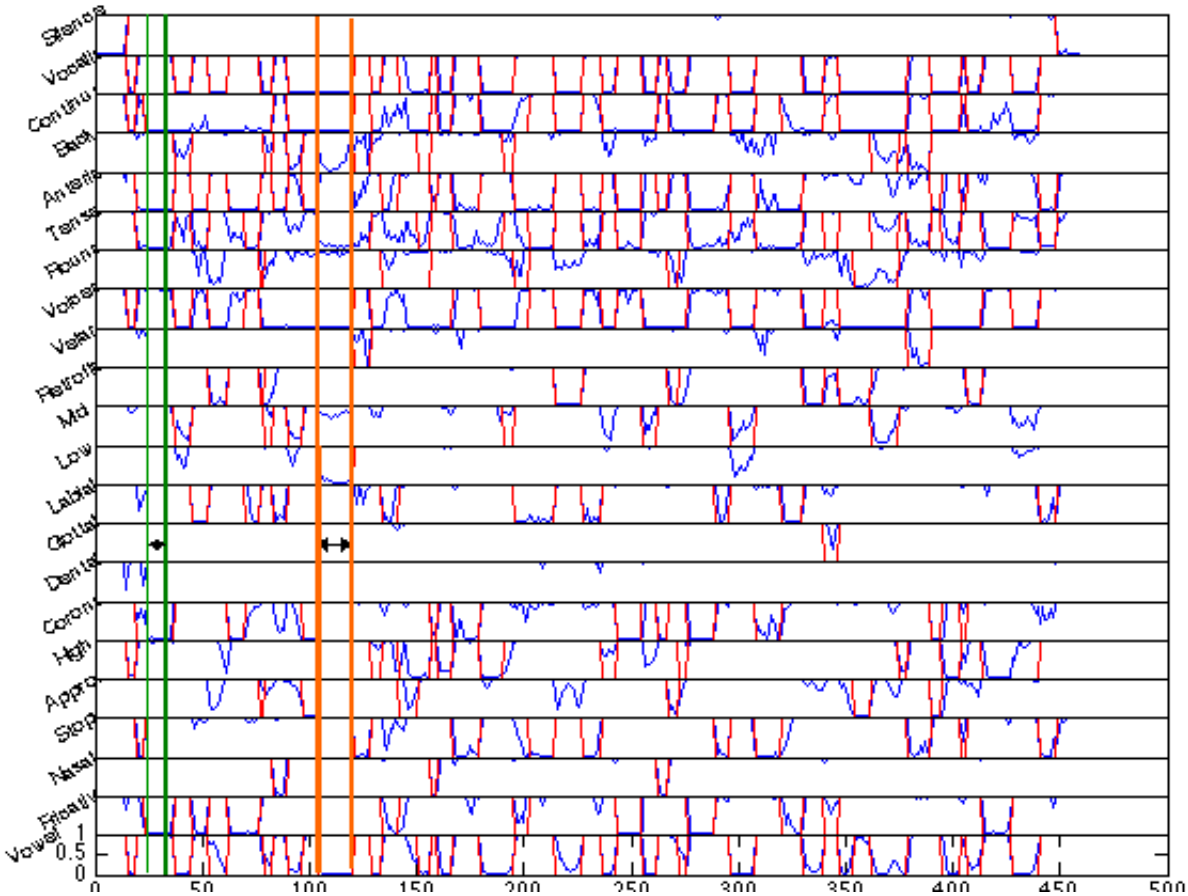


Figure 31: Posteriorgram representation for utterance, ”It suffers from a lack of unity of purpose and respect for heroic leadership.” with TIMIT code fadg0_si649.

There is another interesting observation to mention; traversing across the whole posteriorgram temporally, a relatively high level of correlation and redundancy – in regions where there is not abrupt change in detector outputs – is observed. This implies the stationary behaviour of the 22 dimensional posterior vectors, and verifies the fact that these vectors are qualified to be segmented via segmentation algorithm, and divided into stationary segments. Two relatively stationary regions between two green lines and two orange lines are shown in Figure 31. The MATLAB script to plot this posteriorgram is enclosed in Appendix A.

8.4 Acoustic Segmentation

As described in chapter 4, the segmentation process corresponds with placing boundaries in certain locations along the speech signal, and hence divide the signal into several parts. In this section we will first describe how the termination parameters were selected. We will then answer the following questions:

- Is there observed any consistency in segmentation results?
- Is there any connection between the phonemic segmentation and the acoustic segmentation?
- Is there any connection between the acoustic events observed from the spectrogram representations and the acoustic segmentation results?

Recall from chapter 4, the number of segments is controlled by either over-segmentation factor or average distortion threshold value. However, the phonemic labelling is not available in real applications, and hence employing the over-segmentation factor as a termination criterion is not practical. On the other hand, by defining the threshold to be the stop criterion, the statistical properties inside the segments would roughly be constant and the stationary behaviour of the resulted segments is guaranteed. So the main goal is to find on a “reliable” threshold factor.

The segmentation algorithm is a script written in C, and coupled with visualization and analysis software Praat. So it is possible to visually inspect the segmentation results by comparing them to both manual phonemic segmentations and spectral transitions of the speech signal represented in the form of spectrogram. The more the correspondence between the acoustic results and both of the manual segmentation results and spectrogram representations, the better the algorithm performs its job. However, it is not straightforward to

select the threshold value by performing visual inspection of segmentation results, as there is no high level of consistency in acoustic segmentations. In some cases, it was observed that a lower value for threshold was a better choice and for instance resulted in a better match to the spectral change observed in spectrogram, while in some other cases the opposite seemed to be true. For this reason, we adopt an average strategy, in which the segmentation algorithm is applied to 10 utterances, as shown in Table 3. The first row in this table represents the number of phonemic segments for each utterance. Other rows show the number of acoustic segments corresponding to threshold values from 0.002 to 0.006. In all cases, the over segmentation factor was set equal to 3, which is a relatively high value and makes the over-segmentation factor ineffective. [3] Values in parentheses correspond to the ratio between number of phonemic and acoustic segments for each utterance. An average ratio around 1,5 would be a reliable choice, as theoretically some articulatory features such as stops (found in /p/, /t/, /k/, /b/, /d/ and /g/) or fricatives (found in /ch/, /dh/, /f/, /hh/, /jh/, /s/, /sh/, /th/, /v/, /z/) are characterized by two distinctive events – release and closure – and should accordingly be segmented in two segments.

According to Table 3, the threshold value equal to 0.004 yields an average ratio of 1,59 and can be regarded as a reliable choice. This was the threshold value employed to segment the whole database (both training set and testing set)

[3] Note that there is an upper limit for over segmentation factor, as a large over-segmentation factor (larger than 5) would result in segments smaller than a frame (i.e. smaller than 10 msec). So the number of possible solutions will exceed the size of the distortion matrix and this, in turn, would give unreliable segmentation results.

Table 3: The acoustic segmentation results for 10 utterances

Over-segmentation	Threshold	Prompt	mrtj0_si2032	mjrp0_sx225	frjb0_sx77	makb0_sx296	mwgr0_sx76	fjen0_sx327	mgrl0_si867	mjfh0_si1107	fdfb0_si1318	mbma0_sx412	Average Ratio
		# Phones	25	23	38	30	34	62	54	31	51	36	
3	0.002	# Acoustic Segments	75 (3)	56 (2,43)	114 (3)	90 (3)	102 (3)	186 (3)	162 (3)	93 (3)	153 (3)	108 (3)	2,94
3	0.003		74 (2,96)	43 (1,86)	78 (2,05)	70 (2,33)	76 (2,23)	143 (2,3)	101 (1,87)	80 (2,58)	96 (1,88)	70 (1,94)	2,18
3	0.004		54 (2,16)	38 (1,65)	54 (1,42)	50 (1,66)	53 (1,55)	102 (1,64)	72 (1,33)	60 (1,93)	74 (1,45)	53 (1,47)	1,59
3	0.005		44 (1,76)	27 (1,17)	44 (1,15)	41 (1,36)	44 (1,29)	81 (1,30)	60 (1,11)	49 (1,58)	61 (1,19)	45 (1,25)	1,31
3	0.006		38 (1,52)	24 (1,04)	39 (1,02)	35 (1,16)	38 (1,11)	68 (1,09)	52 (0.96)	42 (1,35)	52 (1.01)	40 (1,11)	1,13

In order to answer to the questions presented at the beginning of the section, we will visually inspect the segmentation results corresponding to an excerpt of several utterances. These are provided in Figure 32 to Figure 39. In each figure, the first part represents the speech waveform of the signal in time domain, the second part is the spectrogram representation, the third part is the phonemic segmentation of the signal and the fourth part corresponds to the acoustic segmentation resulted from the acoustic segmentation algorithm.

Comparing Figure 32 and Figure 33, in both figures the phone **/k/** is segmented into two acoustic segments. This is as desired: these segments correspond to occlusion and release phases of the stop (plosive) feature existing in phoneme **/k/**.

Comparing the vowel **/iy/** in these figures, we see that acoustical segmentation is handled differently. Although the spectral evolution in both **/iy/** phones is almost the same, the faster speech and shorter temporal duration of **/iy/** in Figure 32 has resulted in just one acoustic segment, while relatively slower speech and longer duration of **/iy/** in Figure 33 has led to two segments.

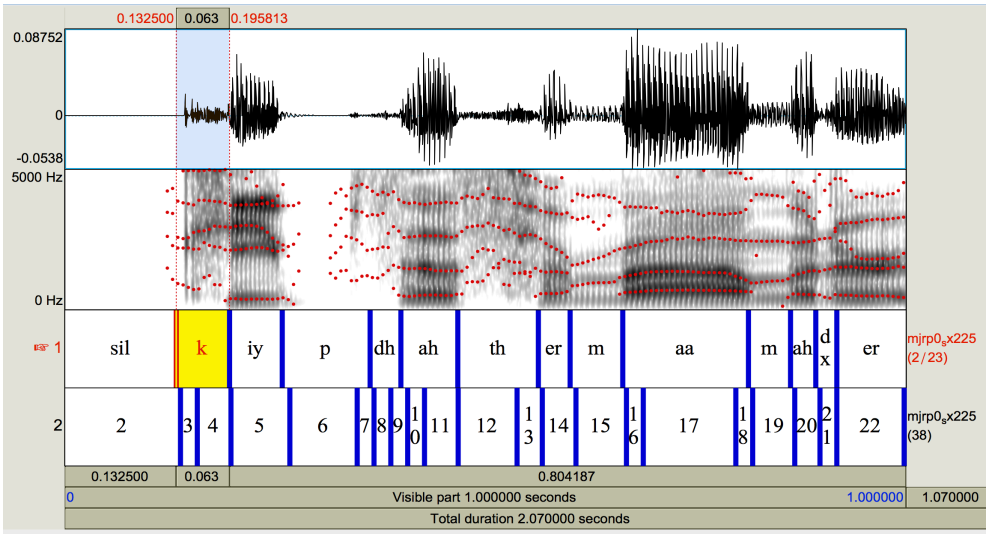


Figure 32: Speech waveform, spectrogram representation, phonemic and acoustic segmentations of the utterance "Keep the thermometer under your tongue!" with TIMIT code mjrp0_sx225

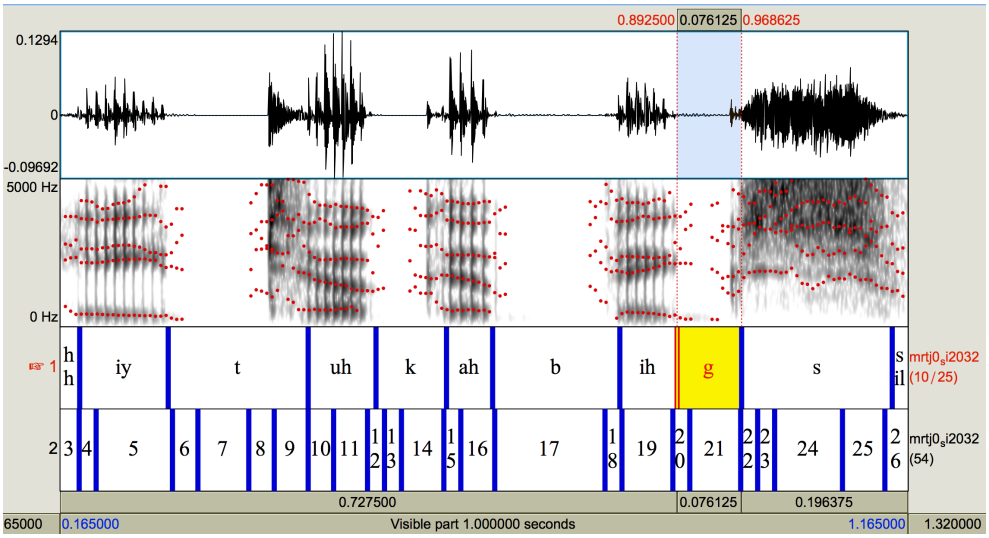


Figure 33: Speech waveform, spectrogram representation, phonemic and acoustic segmentation of the utterance "He took a big swig of his drink." With TIMIT code mrtj0_si2032

In Figure 34, the spectral change in **/ay/** (specially in second formant) is resulted in two acoustic segments. For this utterance, the segmentation algorithm does not seem to do its job reasonably. For instance, despite the gradual spectral change of the phone **/ae/**, it is divided into three segments. Nevertheless, it is important to remember that the threshold parameter is selected from an average strategy among 10 speech utterances. So some degree of inconsistency is expected.

In Figure 35, comparing the spectrogram and the acoustic segmentation results for **/k/**, we can relate the segmentation behaviour to the high degree of spectral change corresponding to that interval in the spectrogram.

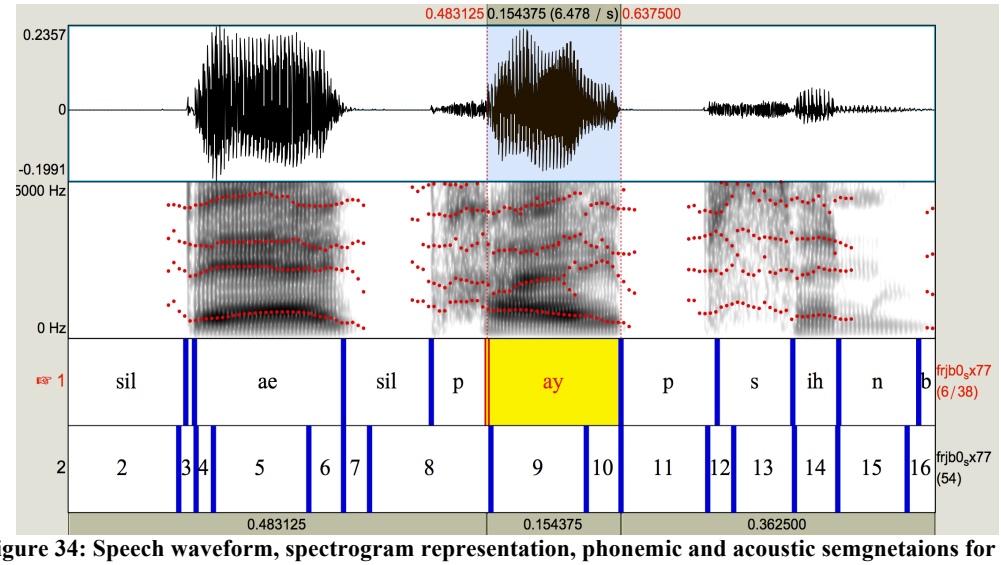


Figure 34: Speech waveform, spectrogram representation, phonemic and acoustic semgnetaions for the utterance "Bagpipes and bongos are musical instruments." With TIMIT code frjb0_sx77

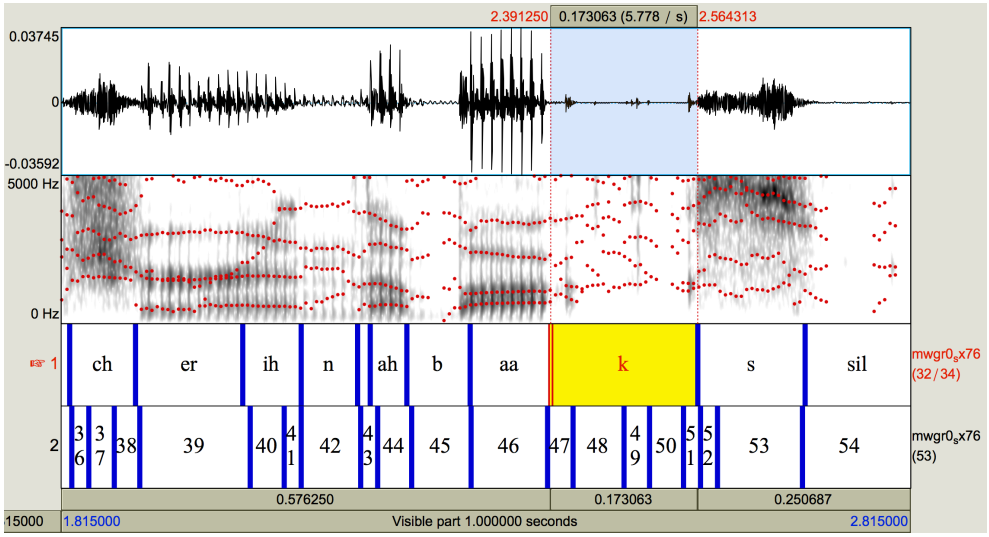


Figure 35: Speech waveform, spectrogram representaiton, phonemic and acoustic segmentation of the utterance "Gently place Jim's foam sculpture in the box." With TIMIT code mwgr0_sx76

Comparing Figure 36 and Figure 37, we see a similar acoustical segmentation pattern for **/w/** followed by **/iy/** in both utterances. Here**, /w/** in both is divided into two acoustic segments with outer boundaries exceeding the phonemic boundaries, and **/iy/** in both contains one acoustic segment with boundaries positioned inside the phonemic boundary. Comparing the acoustic segmentation and the spectrogram representation of the interval **/w//iy/**, one can say that the acoustic boundaries and the temporal spectral change roughly confirm each other.

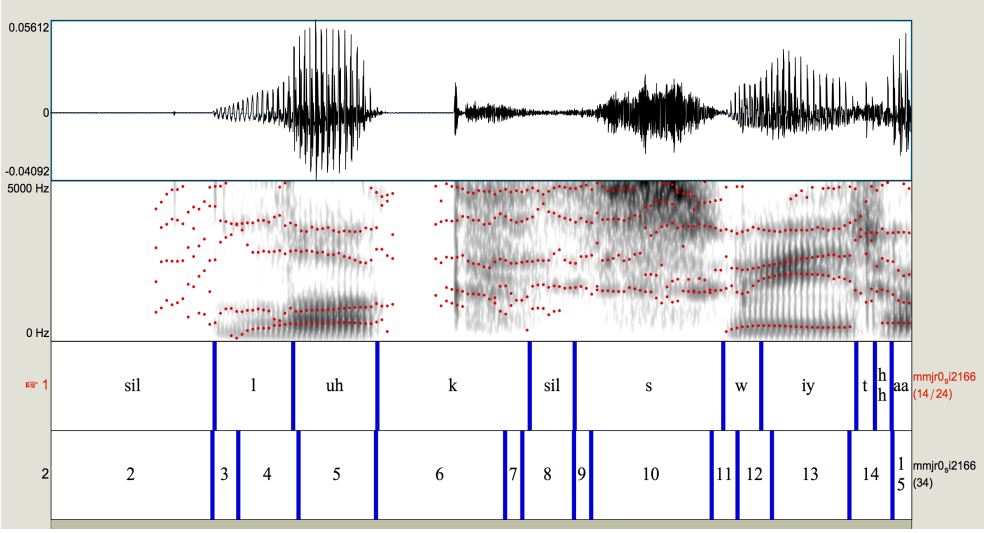


Figure 36: Speech waveform, spectrogram representation, phonemic and acoustic segmentation of utterance "Look, sweetheart, some fool was." with TIMIT code mmjr0_si2166

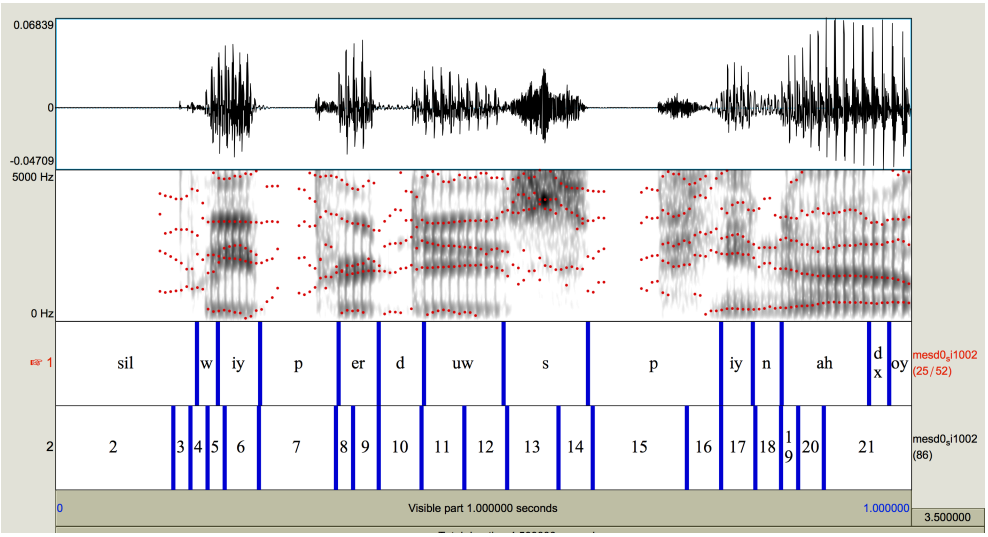


Figure 37: Speech waveform, spectrogram representation, phonemic and acoustic segmnetaiton of an excerpt of the utterance "We produce peanut oil,.." with TIMITcode mesd0_si1002

The two **/p/** phones in Figure 37 also show identical behaviour; both are divided into two segments corresponding to the closure and release phases of the plosive feature, respectively. However, for the first **/p/**, the segmentation settings has led to misplacement of the closure segment (segment no. 8). The boundary misplacement could also be justified due to the coarticulatory effects caused by the surrounding phones.

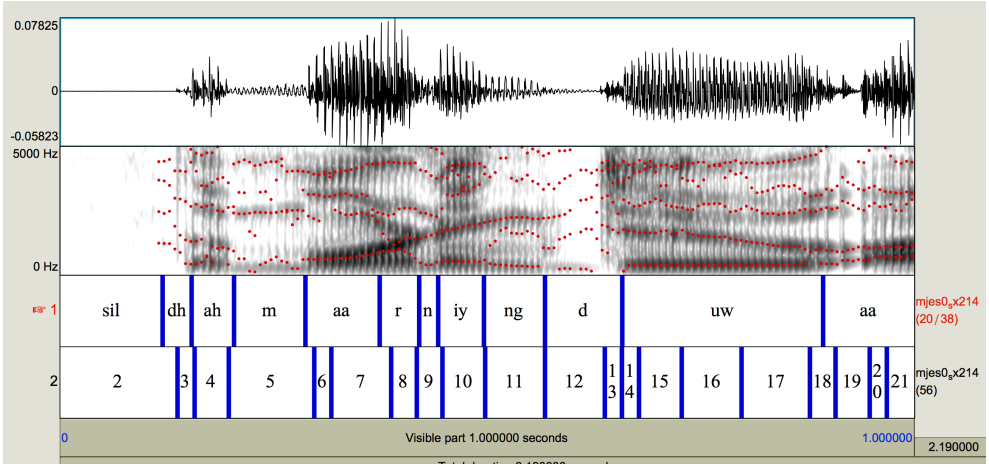


Figure 38: Speech waveform, spectrogram representation, phonemic and acoustic semgentatoin of the utteracne "The morning dew on the spider..." with TIMIT code mjes0_sx214

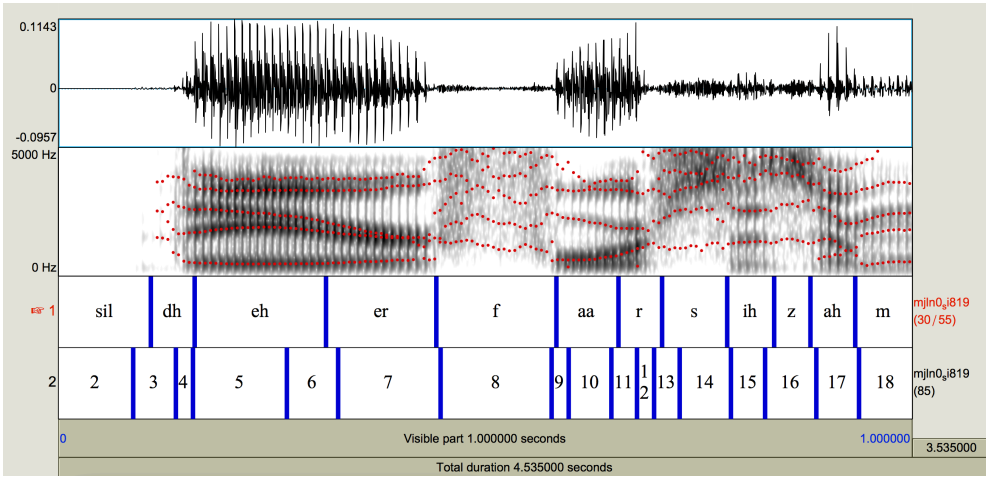


Figure 39: Speech waveform, spectrogram representation, phonemic and acoustic semgentatoin of the utteracne "There, forces are more latent than in electricity, and less than in magnetism." with TIMIT code mjrn0_si819

Comparing the first **/aa/** in Figure 38 and **/aa/** in Figure 39, we can see consistency in acoustic segmentation behavior: in both **/aa/** is divided into a small segment followed by a large one. This is confirmed by looking at the spectrogram representations corresponding to **/aa/**: there is a fairly stable formant pattern in initial part of **/aa/**, while the second segment has formant transitions.

Looking at second **/aa/** in Figure 38, we see that this is treated differently. However, in this case we have another triphone combination **aa(uw,oh)** (the phone **/aa/** proceeded by **/uw/** and followed by **/oh/**) and hence a different type of context dependency exist. Thus, showing another segmentation behaviour is not far from our expectations.

As conclusion, we can say that comparing the acoustic segmentation results and the acoustic events observed from the spectrogram representation, nearly all acoustic segments have one property in common: The acoustic boundaries are more or less positioned in places where there exists spectral change in the spectrogram. This confirms the stationary spectral content inside each segment. Comparing the phonemic and the acoustic segmentation results, we can say that for those phonemes containing stop and fricative feature, there exist more or less two acoustic segments.

Finally, we should note that customizing the segmentation parameters for each individual utterance would obviously result in segments with more reasonable characteristics. However, in this work, it was desired to apply one threshold value to all the utterance samples, at the expense of reduced acoustic consistency.

The segmental centroids were computed through the acoustic segmentation script. However, it was not straightforward to access them directly. Therefore, the MATLAB script Centroidfinding.m was created. This is enclosed in Appendix B.

8.5 Data Clustering

After segmentation, the obtained segments are fed into the k-means algorithm in order to be clustered into k clusters. The optimal number of clusters is a question to be answered in higher recognition levels, after concatenating models and generalizing them to generate higher-level linguistic models such as phonemes, and evaluating their performance. In this work, 128, 256, 512 and 1024 clusters are experimented.

It is worth noting that the number of clusters is first and foremost associated with the parameters defined in segmentation algorithm, namely threshold and over segmentation factors, described in section 4. Changing these factors will lead to different subdivision of database, and subsequently different number of segments. For instance, a low threshold would result in generating more segments, with relatively more stationary characteristics.

The MATLAB script created for k-means clustering is enclosed in Appendix B. The clustering procedure was explained in section (5.1). However, in later step of training HMM models, it turned out that the created k-means script was not able to perform its job properly. For instance, no threshold value could be find to result in 128 clusters. So the built-in MATLAB function k-means was used instead.

8.5.1 Labeling Assignment

After clustering, all segments belonging to the same cluster are labeled by the same cluster label. The procedure was described in section (5.2). In this step, it is attempted to inspect the obtained results to answer the following questions:

- Is there observed any consistency in labeling?
- Is there any relationship between the labels and the phonemic transcriptions?

In this manner, the relabeled files for several utterances together with the corresponding phonemic segmentation – in case of 512 cluster system – were studied. It was observed that almost all acoustic segments corresponding to phonemic segment **/sil/** at the start and end of the utterances are tagged by the same label (L316). This is as expected, as these segments have the same characteristic and hence should be labeled identically. So, the answer to the first question presented above is positive.

Another observation was that a relatively high number of acoustic segments corresponding to the phonemic segment **/iy/** were labeled by L72. It was also observed that several of the acoustic segments corresponding to phonemic segment **/p/** were labeled by L499. This was the case for many of acoustic segments corresponding to **/m/**, which were labeled by L164. So, the answer to the second question presented above is also positive.

8.6 Acoustic modeling and HMM-level recognition

Up to this point, we have been able to annotate the database by a new set of labels. Now all the pieces are in place to start training the HMM models and perform decoding over the test database.

Two experiments are carried out in which various number of HMM models (128, 256, 512, 1024) using single, two and three component Gaussian mixture distributions are investigated. The first experiment involves initialising and training HMM models, decoding the test dataset and analysing the recognition results compared to the original transcriptions. The second experiment deals with initialising HMM models, automatic time alignment of training transcriptions, training HMMs with the obtained transcriptions from the forced alignment step, decoding the test dataset and evaluating the results. For each, we need to define a unique initialisation and training strategy.

In these experiments, the data file is referred to files containing feature vectors for each individual utterance, and the label files refer to files containing transcriptions.

The experiments are best described by going through the involved processing steps.

8.6.1 The first experiment: Recognition

The first task is about decoding the unseen data by HMM models trained by re-labeled training data. The schematic diagram of the steps involved in this task are provided in Figure 40. The modules are shown in blue and the required files are represented in orange. The shell script firstApproach.sh to perform the experiment is enclosed as Appendix D. The steps are described individually in three subsections, as described below.

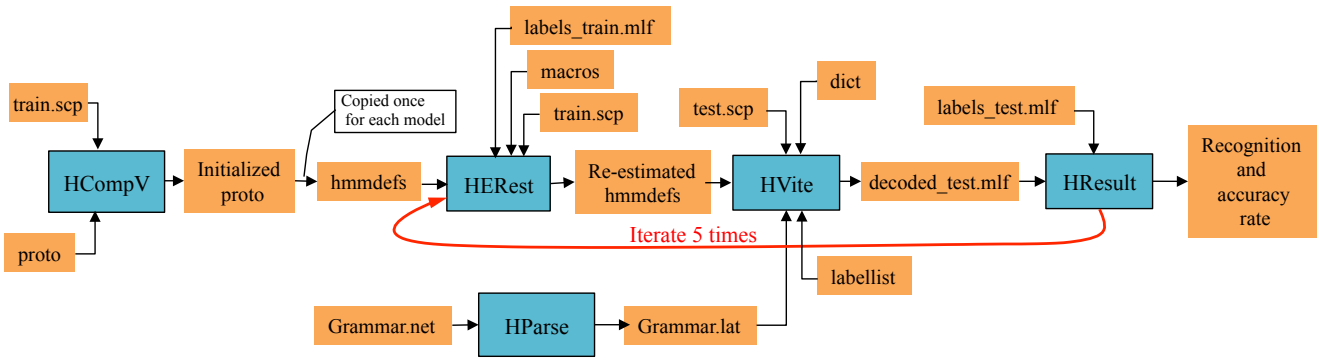


Figure 40: Schematic diagram of the recognition algorithm

1. Initialisation

In order to initialise HMMs, the first step is to create a HMM prototype model, as described in (7.1). This is a single file created according to HMM definition language standards. The prototype models employed in this work are provided in Figure 41. These correspond to (a) single, (b) two-component (with 0.3 and 0.7 Gaussian mixture weights) and (c) three-component (with 0.3, 0.5 and 0.2 Gaussian mixture weights).

~o <VecSize> 23 <FBANK> ~h "proto" <BeginHMM> <NumStates> 3 <State> 2 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <TransP> 3 0.0 1.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 <EndHMM> (a)	~o <VecSize> 23 <FBANK> ~h "proto" <BeginHMM> <NumStates> 3 <State> 2 <NumMixes> 2 <Mixture> 1 0.3 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.00.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <Mixture> 2 0.7 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <TransP> 3 0.0 1.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 <EndHMM> (b)	~o <VecSize> 23 <FBANK> ~h "proto" <BeginHMM> <NumStates> 3 <State> 2 <NumMixes> 3 <Mixture> 1 0.3 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <Mixture> 2 0.5 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <Mixture> 3 0.2 <Mean> 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 <Variance> 23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 <TransP> 3 0.0 1.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 <EndHMM> (c)

Figure 41: Three prototype models (a) single Gaussian, (b) two-component GMM (c) Three-component GMM

Figure 42 shows the corresponding model topology for HMM prototypes represented in Figure 41.

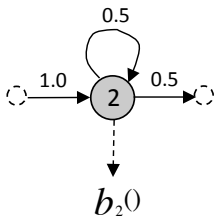


Figure 42: the initial model topology

Now the created HMM prototype model is initialized by all available training data, using HCompV:

```
$ HCompV -A -D -T 1 -C config -f 0.01 -m -S train.scp -M hmm0 proto
```

where proto is the HMM prototype file. The options employed are as follows.

-A: Causes the current command line arguments to be printed.
-D: Used to display configuration settings.
-T: Enables progress-reporting option.
-C: Used to specify the configuration file name.
-S: Used to specify a script file containing all training data filenames (train.scp).
-M: Specifies the output directory (hmm0)
-f: Specifies variance floor macro (0.01)
-m: Used to make the means updated. (The default is to only update the variances)

The initialized proto file is now stored in folder hmm0. There is also created another file called vFloors in the same folder, containing a vector with components equal to 0.01 times the global variance. This vector sets a lower limit for the estimated variances in subsequent steps, and prevents variances to become too small.
Now for each required HMM model, the initialized prototype model is copied once into a file known as HMM definition file. For instance, in case of 128 cluster labels, the HMM definition file would contain 128 repetitions of the initial estimate model. After creating the HMM definition file, some undesired lines should be removed and also the HMM names should be edited such that they correspond to the segment labels.

At next step, the variance floor macro file called macros is created (manually), consisting of the first three lines of the HMM definition file followed by the vector found in the file vFloor, as shown in Figure 43.

```
~o
<STREAMINFO> 1 23
<VECSIZE> 23<NULLD><FBANK><DIAGC>
~v varFloor1
<Variance> 23
 9.390770e-02 1.539757e-01 1.732059e-01 2.056837e-01 2.210912e-01
 2.131917e-01 1.903656e-01 1.674087e-01 1.577063e-01 1.505229e-01
 1.469737e-01 1.400415e-01 1.284006e-01 1.264015e-01 1.244968e-01
 1.205322e-01 1.235889e-01 1.172380e-01 1.084581e-01 1.026291e-01
9.580246e-02 8.806549e-02 8.376861e-02
```

Figure 43: The variance floor macro file

Before start training HMM models, the last file we need to create is the master label file *labels_train.mlf*, containing the transcriptions of all training files. It is created by invoking HLEd as

```
HLEd -l '*' -i labels_train.mlf dummy.led LABELSTRAIN/*
```

Where dummy.led is an empty script, meaning that no edit is subjected to label files. LABELSTRAIN is the directory containing the label files. The following options are used in this invocation.

-l: Causes a label file named xxx to be prefixed by the pattern "*/xxx" in the output MLF file.

-i: Used to specify the resulted MLF (labels_train.mlf).

2. Training

Now we have all tools available to start training the HMM models. The tool used for training is HERest. In the first iteration, it is run as follows

```
HERest -A -D -T 1 -C config -I labels_train.mlf -t 150.0 100.0 500.0 -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 labellist
```

Where ‘labellist’ is the file containing HMM model names. The applied options are:

-A: Causes the current command line arguments to be printed.

-D: Used to display the configuration settings.

-T: Enables the progress-reporting option.

-C: Addresses the configuration file.

-I: Addresses MLF file containing the training transcription files.

-t: Sets the pruning thresholds. (150 100 500)

-H: Loads the HMM definition files. (hmmdefs)

-M: Specifies the directory to store the trained models (hmm1)

3. Decoding

After that the models are trained, decoding can be performed. This is done by invoking the programming tool HVite from the library module HRec. Decoding is performed by a set of trained HMM models, a pronunciation dictionary and a lattice file, as described as below.

- The pronunciation dictionary: In HMM-level recognition, the dictionary contains an entry for all segment labels, such that the segment label and its pronunciation are equivalent. The dictionary file corresponding to 128 segmental units is shown in Figure 44

```
L1  L1
L2  L2
L3  L3
.
.
.
L127  L127
L128  L128
```

Figure 44: The dictionary content for 128 segmental units

- The lattice file: Another file that we need to perform decoding is a lattice file. This is a text file of format HTK Standard Lattice (SLF) created by invoking the tool Hparse, as

```
Hparse Grammar.net Grammar.lat
```

where Grammar.net is a text file containing the grammar rules to specify what kind of inputs are accepted by the system. The suitable grammar corresponding to 128 segmental units is provided in Figure 45(a). It consists of a variable followed by a regular expression. The vertical lines in the right hand side of the variable denote alternatives and the angle braces in the expression denote repetitions. The corresponding HMM-level network is illustrated in Figure 45(b). The network is structured by a simple loop, meaning that each segmental unit may equiprobably follow all other units. The lattice file corresponding to the provided network is shown in Figure 46.

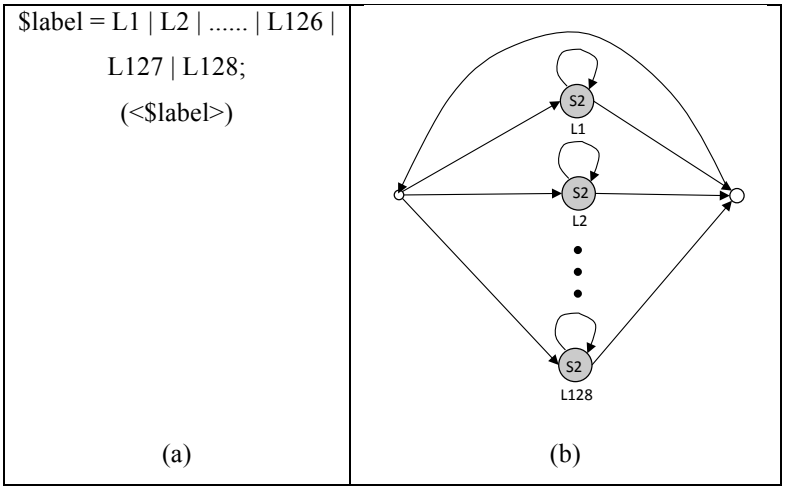


Figure 45: (a)The grammar file and (b) The HMM-level network structure corresponding to the 128 units system.

```
N=131  L=385
I=0    W=L128
I=1    W=!NULL
I=2    W=L127
I=3    W=L126
I=4    W=L125 .
.
.
.
I=129  W=!NULL
I=130  W=!NULL
J=0     S=1    E=0
J=1     S=130  E=0
```

Figure 46: The lattice file corresponding to the 128 units system.

Now, all the required information to perform decoding are available. At the first iteration, HVite is invoked as

```
HVite -A -D -C config -w Grammar.lat -H hmm1/hmmdefs -H hmm1/macros -S test.scp -i decodedresult1.mlf  -p -30.0 dict labellist
```

Where ‘dict’ is the HMM-level dictionary and ‘labellist’ contains all HMM model names. In this invocation the following options are employed:

-A: Causes the current command line arguments to be printed.
-D: Is used to display configuration settings.
-T: Enables progress-reporting option.
-w: Specifies that the recognition is performed from a network.
-H: Loads the HMM model. It should be re-used to load macros. (hmmdefs and macros)
-C: Used to specify the configuration file. (config)
-S: Used to specify the script file containing all the testing data filenames. (test.scp)
-i: Specifies the MLF filename to record the resulting transcriptions. (decodedresult1.mlf)
-p: Specifies the word insertion probability (-30.0)

Observing the obtained MLF after decoding, we see that the boundary locations are not changed. However, after each iteration some of the labels are changed.
Figure 47 shows the recognition results after first and second iteration for 128 segmental units with two component mixture distributions. Those segments where the labels are changed are highlighted.

0 1800000 L104	0 1800000 L104
1800000 2600000 L25	1800000 2600000 L25
2600000 3600000 L6	2600000 3600000 L6
3600000 3800000 L98	3600000 3800000 L98
3800000 4500000 L104	3800000 4500000 L104
4500000 5500000 L93	4500000 5500000 L93
5500000 6300000 L12	5500000 6300000 L12
6300000 7600000 L45	6300000 7600000 L45
7600000 7900000 L8	7600000 7900000 L8
7900000 9200000 L88	7900000 9200000 L88
9200000 9600000 L117	9200000 9600000 L117
9600000 11500000 L112	9600000 11500000 L112
11500000 12200000 L104	11500000 12200000 L104
12200000 12500000 L12	12200000 12500000 L5
12500000 15300000 L104	12500000 15300000 L104
15300000 17000000 L28	15300000 17000000 L28
17000000 17900000 L6	17000000 17900000 L6
17900000 18500000 L116	17900000 18500000 L116
18500000 19900000 L85	18500000 19900000 L85
19900000 22700000 L81	19900000 22700000 L81
22700000 24600000 L6	22700000 24600000 L35
24600000 26600000 L85	24600000 26700000 L85
26600000 27100000 L12	26700000 27100000 L12
27100000 28000000 L104	27100000 28000000 L104

Figure 47: The recognition results corresponding to the utterance mmjr0_si2166

4. Evaluation

In this step the performance of the recognizer is evaluated. This is done by the tool HResults. For the first iteration it is invoked as follows:

```
HResults -A -D -I labels_test.mlf labellist decodedresult1.mlf
```

This tool compares the obtained results after decoding (stored in decoded_test.mlf) with the reference label files (stored in labels_test.mlf) and reports the differences in a text file.

5. Iteration

Now we iterate steps 2, 3 and 4 four more times. The recognition and accuracy results are provided in Table 4, Table 5 and Table 6 for single, two and three-component Gaussian mixtures.

Table 4: Recognition and accuracy results in first experiment using single Gaussian models

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	7.98	7.53	11.62	10.70	16.96	15.27	15.12	13.25	15.59	13.33
256	6.44	6.15	8.42	7.74	10.68	9.44	11.13	9.47	11.43	9.45
512	4.19	4.02	5.54	5.09	7.11	6.18	7.60	6.26	7.45	6.20
1024	2.60	2.48	3.58	3.26	5.15	4.41	5.18	3.99	5.29	3.86

Table 5: Recognition and accuracy results in first experiment using two-component mixture models

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	7.98	7.53	11.62	10.70	17.03	15.35	17.40	15.73	17.40	15.69
256	6.44	6.15	8.42	7.74	10.76	9.49	13.38	11.96	12.16	10.67
512	4.19	4.02	5.54	5.09	7.11	6.18	8.16	7.04	8.13	7.04
1024	2.60	2.48	3.58	3.26	5.15	4.41	5.64	4.13	5.53	4.07

Table 6: Recognition and accuracy results in first experiment using three-component mixture models

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	7.98	7.53	11.62	10.70	17.05	15.37	17.46	15.94	17.93	16.42
256	6.44	6.15	8.42	7.74	11.69	10.40	13.73	12.54	12.80	11.53
512	4.19	4.02	5.54	5.09	7.17	6.23	8.25	7.23	8.02	7.09
1024	2.60	2.48	3.58	3.26	5.58	5.02	5.71	4.26	5.49	4.13

In order to facilitate the comparison, the comparison diagrams corresponding to each table are provided in Figure 48, Figure 49 and Figure 50. As can be observed from the figures, reducing the number of models results in higher recognition and accuracy rates. This is expected, because a smaller number of models correspond to smaller number of clusters, which means that there are a higher number of members belonging to each cluster. This, in turn is equivalent to a higher number of training data for training each model. Another observation is that for all configurations, after a specific number of iterations, the recognition result is either reduced or remained stable. This means that increasing the number of training iterations does not necessarily correspond to a better result. After a number of iterations, the models could even become over-trained and hence fail to recognize the unseen data.

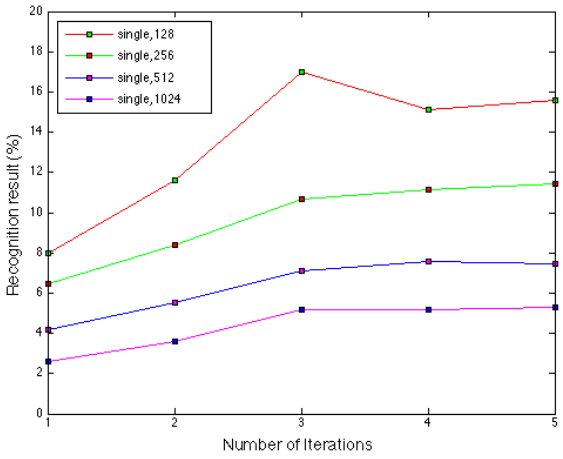


Figure 48: Comparison diagram for first experiment using single Gaussian models

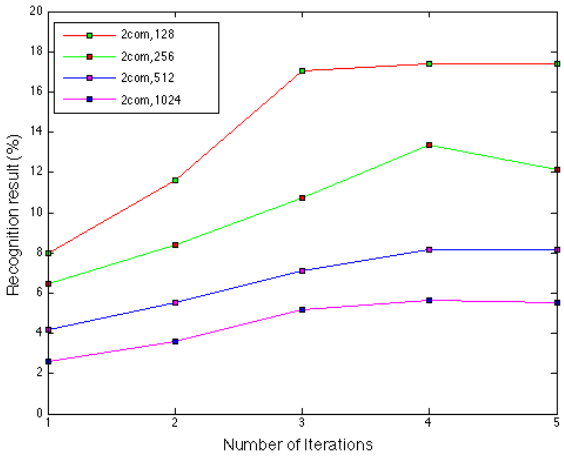


Figure 49: Comparison diagram for first experiment using two-component GMMs

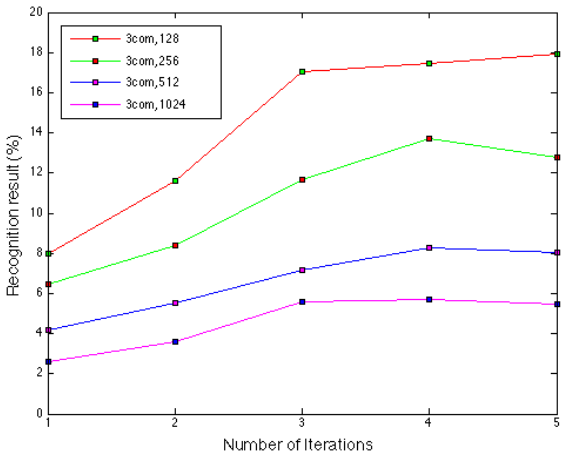


Figure 50: Comparison diagram for first experiment using three-component GMMs

Inspecting the results, another conclusion is that for all systems, up to the third iteration, the recognition and accuracy results are not dependent on the number of mixture components.

8.6.2 The second experiment: Forced Alignment

This experiment deals with employing automatic alignment of segment boundaries corresponding to training data with the speech data, and then using the obtained transcription in training HMMs. The schematic diagram of the experiment is provided in Figure 51. The corresponding script secondApproach.sh is enclosed in Appendix E. This experiment is again divided into three steps, as described below.

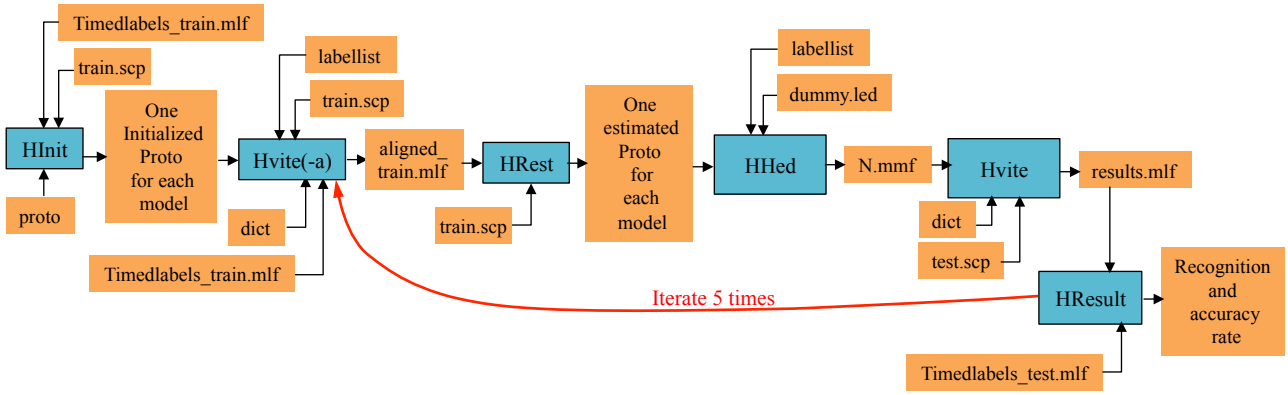


Figure 51: The processing steps regarding the second experiment

1. Initialisation

As described in the section (7.2.2), HInit initializes each model separately. For instance, in order to initialize model L1, HInit is invoked as

```
HInit -I timedlabels_train.mlf -i 2 -l L1 -o L1 -S train.scp -M dir1 proto
```

Where proto is the same prototype HMM model used in the first experiment. Here the following options are employed:

-I: Specifies the MLF file containing the timed aligned segment-level transcription of training set (timedlabels_train.mlf)

-i: Specifies maximum number of Viterbi iterations. (3)

-l: Addresses the label to look for and cut out in the script file. (L1)

-o: Specifies the initialized model name.(L1)

-M: Specifies the directory to store the initialized models. (dir1)

-S: Specifies the script file containing the list of training data filenames. (train.scp)

A typical way to invoke this tool for each model is via a script. The Perl subscript initprotos.pl enclosed in Appendix E does the job for us. After running this script, for each segmental unit a separate file containing the initialised HMM definition is created.

2. Forced Alignment

Now we perform Forced Alignment on training data. For this purpose, HVite is invoked as follows:

```
$ HVite -A -D -T 1 -l '*' -o M  -C config -H N2.mmf -i aligned1.mlf -y lab -a -I timedlabels_train.mlf -S train.scp  -p -30.0 dict labellist
```

where ‘dict’ is the dictionary specified in the first experiment, and ‘labellist’ is the file containing the list of model names. The employed options are as follows:

-A: Causes the current command line arguments to be printed.

-D: Used to display configuration settings.

-T: Enables progress-reporting option.

-l: Addresses the directory to store output files. Using ‘*’ causes a label file named xxx to be prefixed by the pattern "*/xxx" in the output MLF file.

-o: Specifies output level formatting.

-C: Used to specify the configuration file name. (config)

-H: Loads HMM macro model file (MMF). It may be repeated to load several macros.(N2.mmf)

-a: Activated to make HVite Perform forced alignment (instead of –w which was used in decoding)

-S: Used to specify the script file corresponding to the training data. (train.scp)

-i: Addresses the MLF file to store the aligned training transcriptions. (aligned1.mlf)

-p: Sets the word insertion log probability (-30)

-y: Specifies the output file extension (lab)

-I: Specifies the MLF file to be loaded. (timedlabels_train.mlf)

Figure 52 shows the transcriptions corresponding to the utterance mjrp0_sx225 in case of 128 segmental units system before and after first iteration of forced alignment. As we see, in both cases the sequences of the labels are the same, while the segmental time boundaries have changed.

0 1400000 L104	0 1300000 L104
1400000 2000000 L29	1300000 1800000 L29
2000000 2700000 L84	1800000 2500000 L84
2700000 3500000 L24	2500000 3600000 L24
3500000 3700000 L85	3600000 3900000 L85
3700000 4100000 L99	3900000 4000000 L99
4100000 4700000 L76	4000000 4600000 L76
4700000 5400000 L93	4600000 5500000 L93
5400000 5700000 L74	5500000 5600000 L74
5700000 6100000 L59	5600000 5900000 L59
6100000 6700000 L124	5900000 6500000 L124
6700000 8200000 L86	6500000 8900000 L86
8200000 8700000 L124	8900000 9000000 L124
8700000 9000000 L59	9000000 9100000 L59
9000000 9200000 L10	9100000 9200000 L10
9200000 10000000 L44	9200000 10000000 L44
10000000 10200000 L32	10000000 10100000 L32
10200000 10500000 L43	10100000 10300000 L43
10500000 10800000 L32	10300000 10600000 L32
10800000 11300000 L113	10600000 11300000 L113
11300000 12000000 L25	11300000 11800000 L25
12000000 12800000 L46	11800000 12900000 L46
12800000 13700000 L97	12900000 13800000 L97
13700000 14000000 L105	13800000 13900000 L105
14000000 14600000 L46	13900000 14500000 L46
14600000 16000000 L121	14500000 15700000 L121
16000000 17500000 L19	15700000 17100000 L19
17500000 17700000 L36	17100000 17600000 L36
17700000 18100000 L82	17600000 17700000 L82
18100000 18400000 L126	17700000 18600000 L126
18400000 20500000 L104	18600000 20500000 L104

Figure 52: The transcription corresponding to the utterance mjrp0_sx225 before and after performing forced alignment

3. Training

At this step, the training phase starts. Training in this experiment is performed by the programming tool HRest. It is again more efficient to call this tool via a script, as this tool is applied individually to each initialised model. The Perl subscript trainafteralignment.pl

provided in Appendix E is intended to make this done. As said before, the main goal of this experiment is to evaluate the performance of forced alignment. Therefore, here the time-aligned transcriptions obtained from the forced alignment step are used to train the models. As an example, in order to train model L1 it is run as:

```
HRest -T 1 -I aligned1.mlf -i 20 -l L1 -M dir2 -S train.scp dir1/L1
```

Where dir1/L1 indicates that the model to be trained is located in directory dir1. Here the following options are invoked:

-T: Enables progress-reporting option.
-I: Specifies the MLF file containing the label-level transcription of training set accompanied with the segment boundaries. (aligned1.mlf)
-i: Specifies maximum number of iterations. (20)
-l: Addresses the label to look for (and cut out) in the transcription file. (L1)
-M: Addresses the directory to store the initialized model. (dir2)
-S: Specifies a script file containing the list of training data filenames. (train.scp)

At this time, we collect all HMM definition files into one MMF file. This is performed using HHEd:

```
$ HHEd -w dir2/N2.mmf -d dir2 dummy.led labellist
```

where ‘dummy.led’ is an empty file, and does simply nothing. It means that no edit is subjected to HMM definitions. The file ‘labellist’ is containing all the HMM model names, one per line. The following options are used in this invocation:

-w: Addresses the MMF file which all HMM definitions are to be stored in. (N2.mmf)
-d: Specifies where to look for each HMM definition file. (dir2)

4. Decoding

Now we perform decoding. In the first iteration, it is invoked as follows:

```
HVite -A -D -T 1 -C config -w Grammar.lat -H N2.mmf -S test.scp  -i results1.mlf -p -30.0
dict labellist
```

The recognized MLF is stored in the file results1.mlf. The options are the same as those used at decoding step in the first experiment.

5. Evaluation

The recognition performance is computed via HResults. For the first iteration, it is run as follows

```
HResults -A -D -I timedlabels_test.mlf labellist results1.mlf
```

6. Iteration

Now we iterate steps 2, 3 and 4 and 5 iteratively four more times. The recognition and accuracy results are provided in Table 7, Table 8 and Table 9 for single, two and three-component Gaussian mixtures.

Table 7: Recognition and accuracy results in second experiment using single Gaussian models

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	16.17	14.75	16.47	14.99	16.48	15.00	16.50	14.96	16.48	14.96
256	12.03	10.94	12.19	11.01	12.29	11.10	12.36	11.12	12.39	11.13
512	7.83	7.05	7.45	6.56	7.43	6.54	7.43	6.53	7.44	6.54
1024	4.84	4.11	4.83	4.03	4.87	4.05	4.83	3.98	4.83	3.99

Table 8: Recognition and accuracy results in second experiment using two-component GMMs

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	16.12	14.80	16.00	14.59	15.85	14.44	15.74	14.36	15.66	14.29
256	11.56	10.45	11.42	10.25	11.38	10.13	11.36	10.08	11.32	10.04
512	7.89	7.07	7.90	7.08	7.83	7.06	7.83	6.99	7.86	6.97
1024	4.72	4.13	4.75	4.09	4.53	4.11	4.69	3.99	4.71	4.06

Table 9: Recognition and accuracy results in second experiment using three-component GMMs

Number of units	1st iteration(%)		2nd iteration(%)		3rd iteration(%)		4th iteration(%)		5th iteration(%)	
	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc	Rec	Acc
128	17.23	16.22	17.26	16.25	17.19	16.19	17.18	16.17	17.17	16.13
256	13.11	12.22	13.14	12.22	13.08	12.18	13.01	12.10	12.92	12.03
512	9.07	8.40	9.09	8.39	9.11	8.36	9.07	8.36	9.09	8.37
1024	6.13	5.81	6.12	5.64	6.18	5.42	6.11	5.54	6.14	5.56

In this experiment, we again observe that by reducing the number of segmental units, the recognition and accuracy results are improved. Another observation is that already form the first iteration, the recognition and accuracy results are relatively high (compared to first experiment). However in this experiment iterating does not contribute to better performance: Looking at the results of each system, all results corresponding to all iterations are roughly the same. This is also found from Figure 53, Figure 54 and Figure 55.

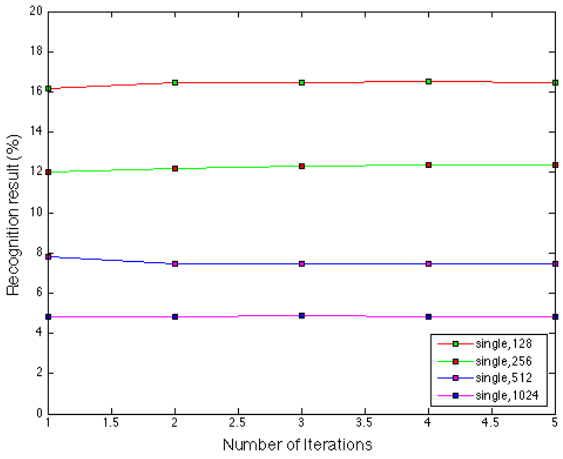


Figure 53: Comparison diagram for second experiment using single Gaussian models

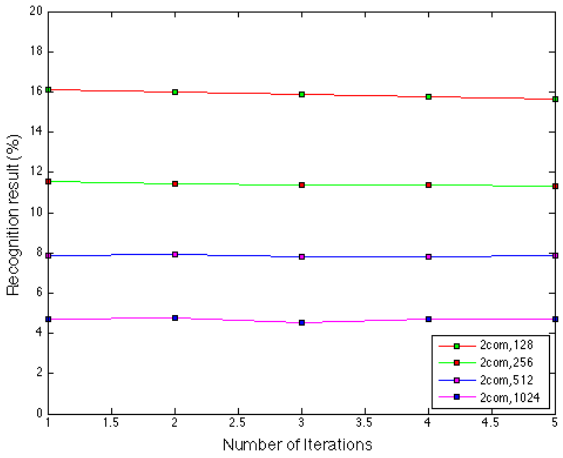


Figure 54: Comparison diagram for second experiment using two-component GMMs

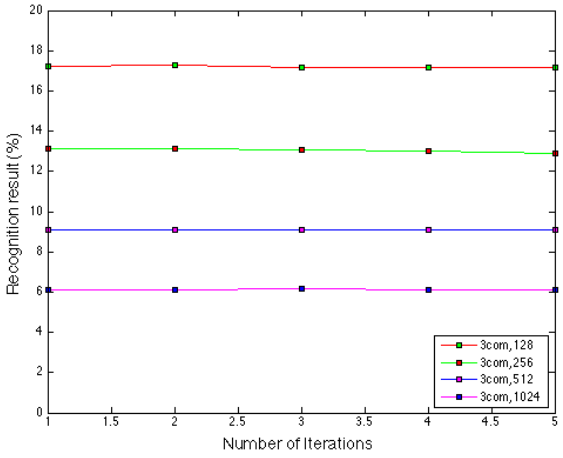


Figure 55: Comparison diagram for second experiment using three-component GMMs

Comparing the results obtained in this experiment with those from the first experiment, one could say that in all cases, employing the forced alignment has improved the performance of the systems. The other observation is that here again, the system with 128 segmental models outperforms other systems. It is also observed that using three-components, compared to single and two-component Gaussian models, has resulted in better performance.

8.7 Summary

In this chapter the experimental setup, the implementations, the results and discussions corresponding to the experiments performed in the project were covered. Several questions were answered and the results were discussed.

9 Summary, Conclusions and Future work

In this final chapter, a brief summary of the work is presented, and the most important conclusions are outlined. Some suggestions for future work will also be given.

9.1 Summary and Conclusions

In this project we were aimed to design a speech recognizer based on a new collection of sub-phonemic units. These units were found during several steps by employing several machine learning algorithms: After the extraction of MBE feature vectors followed by STC generation and acoustic-phonetic detection of all STC vectors, the posterior probabilities corresponding to existence/absence of a set of 22 articulatory features such as nasality, fricativity, vocalisation, glottalisation etc. were stacked in form of long vectors called posteriori supervectors. These vectors constructed the input data to acoustic segmentation module. At segmentation step, the signals were segmented in such a way to result in minimum segmental distortion. The segments were represented by their centroids, known as representative vectors. These representatives were input to a clustering algorithm by which the vectors with the most similar acoustic-phonetic characteristic were stamped by the same label, corresponding to the cluster label they were belonging to. Hereby, the whole database was re-labeled by the labels corresponding to the cluster labels. The final step, which was the main focus of this study, was to build a set of acoustic models – one model for each label – explore their properties and investigate their performance on a unit-level speech recognizer.

In order to evaluate the performance of the models, two different approaches were experimented, namely forced alignment and decoding. Both experiments were performed on 128, 256, 512, and 1024 acoustical units systems. Results show that applying forced alignment on training transcriptions improves the performance of all systems. The best results were achieved for the 128 units system. The recognition system designed in this work operates with error rates that are relatively high to be used in practice. The poor recognition results are however to some degree expected, this because we have employed a high number of acoustic models with relatively low amount of training data.

In next section some ideas to be investigated in future work are suggested.

9.2 Future Work

The experiments introduced in this work correspond to building a unit-level recognition system. After HMMs each corresponding to a speech unit are trained and tested, the next step is to generalize these HMMs by connecting them together in sequence, to construct vocabulary. Due to time constraints, it was not possible to design a higher-level recognition system and the question of generalizability of the models remained unanswered. Therefore, first of all, it is necessary to perform further experiments and create the final recogniser machine, before making a conclusion and argue about the acceptance or rejection of the studied approach.

From the articulatory point of view, both the window shift and the window length should be adapted to the variation of the articulators and the change in the vocal tract shape [12, p.164]. In this work, these were assumed to be identical for all articulators. Costumizing them could be subject for further exploraiton.

Regarding the segmentation algorithm, choosing a threshold for the whole data set as a stop criterion is a challenging task, since characteristics of each signal is different from the others. The other termination alternative was to set the stop condition relative to the manual phonemic labels for each utterance. The downside with this approach however, is that the manual labelling is not available in real applications. For these reasons, it would be interesting to define the stop criterion with regard to some other cues in future work.

In case of K-means algorithm, experimenting other initialisation settings for cluster centroids would be of interest.

In case of acoustic modeling, other probability estimators such as tied mixtures were not examined in this work. This could be investigated in future. It would also be interesting to consider higher number of internal states in defining HMM models.

In case of HMM prototype, just three configurations (single, two and three component mixtures) were experimented in this work. This could be optimized in a future work.

Bibliography

[1] Aggarval, K., & Dave, M. (2011). Acoustic modeling problem for automatic speech recognition system: conventional methods (Part I).

[2] Chomsy, N., & Halle, M. (1968). *The Sound Pattern of English.* New York: Harper & Row.

[3] Daumé, H. *A course in machine learning.*

[4] Frankel, J., Wester, M., & King, S. (2007). Articulatory feature recognition using dynamic Bayesian networks.

[5] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., & Pallett, D. S. (1993). *DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus.* Gaithersburg, USA: U.S. Dept. of Commerce.

[6] Gershenson, C. (2001, Autumn). *Artificial Neural Networks for Beginners.* From http://arxiv.org/abs/cs/0308031

[7] Huang, X., Acero, A., & Hon, H. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm and System Development.* Prentice Hall.

[8] Jones, E. R. (2004, December). *An Introduction to neural networks, A white paper.* From Visual Numerics, Inc.: http://www.roguewave.com/DesktopModules/Bring2mind/DMX/Download.aspx?entryid=756&command=core_download&PortalId=0&TabId=607

[9] King, S., & Taylor, P. (2009). Detection of phonological features in continuous speech using neural networks. *Comput. Speech Lang. , 14*, 333-345.

[10] Ladefoged, P. & Johnson, K. (2010). *A course in phonetics.* Boston: Michael Rosenberg.

[11] Ostendorf, M. (1999). Moving beyond the 'beads-on-a-string' model of speech. *In: IEEE ASRU Workshop* , 230-245.

[12] Raj, B., Virtanen, T., & Singh, R. (2012). *Techniques for Noise Robustness in Automatic Speech Recognition.* John Wiley & Sons.

[13] Schwarz, P. (2008). *Phoneme recognition based on long temporal context* . Doctoral thesis, Brno University of Technology, Faculty of Information Technology, Brno.

[14] Rabiner, L. & Juang, B. (1993). *Fundamentals of Speech Recognition.* New Jersey: Prentice Hall.

[15] Schwarz, P., Matejka, P., & Cernocky, J. (2006). Hierarchical structures of neural networks for phoneme recognition. *Proc. ICASSP* , 325 -328.

[16] Siniscalchi, S. M. (2012). Combining speech attribute detection and penalized logistic regression for phoneme recognition. Neurocomputing, vol. 93, pp. 10–18.

[17] Smola, A., & S.V.N, V. (2008). *Introduction to machine learning.*
Svendsen, T., & Soong, F. (1987). On the automatic segmentation of speech signals. *IEEE International Conference on ICASSP '87* .

[18] Svendsen T. (2013) *Lecture notes in TTT4185* (Speech recognition)

[19] Svendsen, T.; Soong, F. K. (1987) *On the automatic segmentation of speech signals, IEEE International Conference on ICASSP '87*

[20] Jang R, Data clustering and pattern recognition, chapter 3,
http://mirlab.org/jang/books/dcpr/dcKMeans.asp?title=3-3%20K-means%20Clustering

[21] Jurafski D.; Martin J. H. (1999) *Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*

[22] Young etal, (2001) *The HTK Book*, Cambridge University Engineering Department

[23] Zhai, C. (2003). *A Brief Note on the Hidden Markov Models (HMMs).*

APPENDIX

APPENDIX A: POSTERIORGRAM REPRESENTATION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Posteriorgram of posterior vectors corresponding to 566th sentence of TIMIT
%test (fadg0_si649)in htk format.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc

[AF,fp,dt,tc,t]=readhtk('/Users/negarolfati/Desktop/Detection/AF_TIMIT/1_state/train_phone_recognizer/pfiles/
test_566.htk');

nFeatures=22;

% the manual labels are extracted from the last column of the files with the
% following format: [referencePrefix, label{i}, referenceExtension]; For
% generating these files, we made use of the labelextraction.sh script
% enclosed after this script.

referencePrefix='/Users/negarolfati/Desktop/Detection/AF_TIMIT/1_state/train_detectors/binlabels/labels_';


label = {'Vowel','Fricative','Nasal','Stop','Approx','High','Coronal','Dental',...
   'Glottal','Labial','Low','Mid','Retroflex','Velar','Voiced','Round',...
'Tense', 'Anterior','Back','Continuant','Vocalic','Silence'};

referenceExtension='.txt';
%%
%plot each detector output saparately
for i = 1:22
   figure(i)
   plot(AF(:,2*i));
   ylabel(label{i});
   xlabel('frame number');
   title('htk\_gram, fadg0\_si649');
   hold on
   filename = [referencePrefix, label{i}, referenceExtension];
   reference=load(filename);
   plot(reference(:,26), 'r');

end

%% Create the posteriorgram


figure
for i=1:22
   h=subplot('Position',[0.08 (i-1)*0.96/22+0.02 0.90 0.96/22]);
   axis([1 459 -10 0.5]);
   plot(AF(:,2*i));
   if (i > 1)
```

```
        set(h,'XTick',[],'YTick',[]);
    end
    ylabel(label{i},'rot',30);
    hold on
    filename = [referencePrefix, label{i}, referenceExtension];
    reference=load(filename);
    plot(reference(:,26), 'r');
end
```

Script: ~/Detection/AF_TIMIT/1_state/LabelExtraction.sh

```
#!/bin/bash

TOP=`pwd`
export BINDIR=/usr/local/bin
featuremap=extract_melbankenergies/data/Marco_AF.map
AFlist=extract_melbankenergies/data/AFlist
GetAFset.pl $featuremap $AFlist

for AF in `cat $AFlist`
do

for i in 566
do

# Convert to htk format
$BINDIR/feacat -i $TOP/train_detectors/trained/$AF/labeledpfiles/test.pfile -ip pfile -o
$TOP/train_detectors/htks/htk_$AF.htk -op htk -sr $i:$i

# save the files containing the binary labels for each detector
$BINDIR/pfile_print -i $TOP/train_detectors/trained/$AF/labeledpfiles/test.pfile -sr $i:$i  >&
$TOP/train_detectors/binlabels/labels_$AF.txt

done
done
```

APPENDIX B: ACOUSTIC SEGMENTAL CENTROID COMPUTATION, CENTROID CLUSTERING AND LABEL ASSIGNMENT

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this script, the following are performed:
%1. The posterior supervectors resulted from acoustic merging and the segment boundaries resulted from the
%acoustic segmentation are imported.
%2. The dimension of posterior supervectors is reduced to 22.
%3. The segmental centroids are computed and clustered via the k-means algorithm.
%4. The labels corresponding to clusters are assigned to relative segments.
%5. The label files are exported both with and without boundary information.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc
close all

% Load data

htkfiles_train = dir('/Users/negarolfati/Desktop/Acsegment/mfcdir/train/*.htk');
htkfiles_test = dir('/Users/negarolfati/Desktop/Acsegment/mfcdir/test/*.htk');

htkpaths_train = (strcat({'/Users/negarolfati/Desktop/Acsegment/mfcdir/train/'},{htkfiles_train.name}))';
htkpaths_test = (strcat({'/Users/negarolfati/Desktop/Acsegment/mfcdir/test/'},{htkfiles_test.name}))';


nTrain = length(htkpaths_train);
files_train = cell(nTrain,1);

for k = 1:nTrain
   files_train {k,1} = htkfiles_train(k,1).name; % store the train list in 'files_train'.
end

nTest = length(htkpaths_test);

nTest = length(htkpaths_test);
files_test = cell(nTest,1);
for k = 1:nTest
   files_test {k,1} = htkfiles_test(k,1).name; % store the test list in 'files_test'.
end

% remove the extension '.htk' from filenames
for k = 1: nTrain
   files_train{k,:} = strrep(files_train{k,1},'.htk','');

end

for k = 1: nTest
   files_test{k,:} = strrep(files_test{k,1},'.htk','');

end


% Now, generate the '.acs' paths. These are the files containing
% time-boundaries of the segments.
for k = 1:nTrain
   acspaths_train{k,1} =
(strcat(('/Users/negarolfati/Desktop/Acsegment/espslabs/train/'),(files_train{k,1}),'.acs'))';
```

```
end

for k = 1:nTest
   acspaths_test{k,1} = (strcat(('/Users/negarolfati/Desktop/Acsegment/espslabs/test/'),(files_test{k,1}),'.acs'))';

end



% Remove the special char # from .acs files, in order to make them loadable via 'load' function

command =  'perl -pi -e "BEGIN{ @ARGV = glob(pop) } s/#//g"
"/Users/negarolfati/Desktop/Acsegment/espslabs/train/*"';
status = system(command);
command =  'perl -pi -e "BEGIN{ @ARGV = glob(pop) } s/#//g"
"/Users/negarolfati/Desktop/Acsegment/espslabs/test/*"'; % Remove the special char #
status = system(command);


% load *.htk files. Store them in first column of data_train and data_test

nAF = 22; % Number of Articulatory Features, in other words the dimension of data points.
data_train = cell(nTrain,6);
for k = 1:nTrain
   data_train{k,1} = readhtk(htkpaths_train{k,1});
   m = data_train{k,1};
   c = zeros(length(m),nAF);
   for t = 1:nAF       %  activation = 1-deactivation, so keep one component from each two.
      c(:,t) = m(:, 2*t-1);
      data_train{k,1} = c;
   end

end
%
data_test = cell(nTest,6);
for k = 1:nTest
   data_test{k,1} = readhtk(htkpaths_test{k,1});
   m = data_test{k,1};
   c = zeros(length(m),nAF);
   for t = 1:nAF
      c(:,t) = m(:, 2*t-1);
      data_test{k,1} = c;
   end

end


% Load *.acs files, store them in second column in cell data_train and data_test
for k = 1:nTrain
   a = load(acspaths_train{k,1});
   data_train{k,2} = a(:,1);
end
%
for k = 1:nTest

   a = load(acspaths_test{k,1});
   data_test{k,2} = (a(:,1));
end
```

```
%%  Compute segmental mean for train data and test data

for k = 1:nTrain
   BoundPos = data_train{k,2}*1e3/10;  % framerate = 10 msec

   % Store time-boundarieds for all segments corresponding to training
   % data in 5th and 6th column of data_train.
   BoundPos(:) = round(BoundPos(:));
   data_train{k,5}=[0;BoundPos(1:end-1,1)];
   data_train{k,6}=BoundPos;
   nsegMembers= zeros(1,length(BoundPos));

   % How many members are present in each segment?
   nsegMembers(1) = round(BoundPos(1));
   for t = 2: length(BoundPos)
      nsegMembers(t) =  BoundPos(t) - BoundPos(t-1);
   end

   nsegMembers(:) = round(nsegMembers(:));

   % Store number of segments for all training data in 4th column of
   % data_train
   data_train{k,4} = nsegMembers;

   % Compute segmental means and store them in 3rd column.
   mean = zeros(nAF, length(BoundPos));
   FrameIdx = 1;
   d = (data_train{k,1})';

   for s = 1 : length(BoundPos)
      mean(:,s) = sum(d(:,(FrameIdx:(BoundPos(s)))),2)/nsegMembers(s);
      FrameIdx = FrameIdx + nsegMembers(s);
   end
   data_train{k,3} = mean;
end

%

for k = 1:nTest
   BoundPos = data_test{k,2}*1e3/10;  % framerate = 10 msec
   BoundPos(:) = round(BoundPos(:));
   data_test{k,5}=[0;BoundPos(1:end-1,1)];
   data_test{k,6}=BoundPos;
   nsegMembers= zeros(1,length(BoundPos));


   % How many members are present in each segment?
   nsegMembers(1) = round(BoundPos(1));
   for t = 2: length(BoundPos)
      nsegMembers(t) =  BoundPos(t) - BoundPos(t-1);
   end

   nsegMembers(:) = round(nsegMembers(:));
   data_test{k,4} = nsegMembers;
   % Compute segmental means
   mean = zeros(nAF, length(BoundPos));
   FrameIdx = 1;
   d = (data_test{k,1})';

   for s = 1 : length(BoundPos)
```

```
        mean(:,s) = sum(d(:,(FrameIdx:(BoundPos(s)))),2)/nsegMembers(s);
        FrameIdx = FrameIdx + nsegMembers(s);
    end
    data_test{k,3} = mean;
end

%% How many mean vectors are generated?
meansize_train = 0;
for k = 1:nTrain
    sentmeansize = (size(data_train{k,3},2));
    meansize_train = sentmeansize + meansize_train;
end

meansize_test = 0;
for k = 1:nTest
    sentmeansize = (size(data_test{k,3},2));
    meansize_test = sentmeansize + meansize_test;
end



%% Now prepare data to be input to cluster algorithm
data= [data_train ; data_test];
files= [files_train; files_test];
meanstore_tot = cell2mat(data(:,3)');
%pause
save('meanstore_tot.mat', 'meanstore_tot');

%[IDX_data] = myKmeans(meanstore_tot,thr_data);

[IDX_data] = kmeans(meanstore_tot',128);
IDX_data = IDX_data';
%% perform label assignment, and export labels files Without
%time boundaries. These to be used in first experiment.
nData = nTrain+nTest;
a = cell(1,nData);

for k = 1:nData
    a{k} = data{k,4};
end

vectorSizes_data=cellfun(@(x) numel(x),a);
idx_data = mat2cell(IDX_data', vectorSizes_data);
%
for k = 1:nTrain
    f = idx_data{k};
    fid =
fopen(strcat(('/Users/negarolfati/Documents/Detection_rerun/AF_TIMIT/1_state/mergedlabels_train/'),(files{k})
,'.lab'),'wt');
    fprintf(fid, [repmat('%g\n', 1, size(f,2)-1) '%g\n'], f.');
    fclose(fid);
end
%

for k = nTrain+1 : nData

    f = idx_data{k};
    fid =
fopen(strcat(('/Users/negarolfati/Documents/Detection_rerun/AF_TIMIT/1_state/mergedlabels_test/'),(files{k}),'
.lab'),'wt');
```

Appendix B

```
    fprintf(fid, [repmat('%g\n', 1, size(f,2)-1) '%g\n'], f.');
    fclose(fid);

end

%%  Export label files with time boundaries. These to be used in second experiment.


for k = 1:nTrain
    idx_train{k}=idx_data{k};
    f = [data{k,5}*10^5 data{k,6}*10^5 idx_data{k}];
    filename =
strcat(('/Users/negarolfati/Documents/Detection_rerun/AF_TIMIT/1_state/timedlabels_train/'),(files{k,1}),'.lab');
    fileID = fopen(filename,'w');
    fprintf(fileID,'%d %d %d\n',f');
    fclose(fileID);
end


for k = nTrain+1 : nData
    idx_test{k}=idx_data{k};
    f = [data{k,5}*10^5 data{k,6}*10^5 idx_data{k}];
    filename =
strcat(('/Users/negarolfati/Documents/Detection_rerun/AF_TIMIT/1_state/timedlabels_test/'),(files{k,1}),'.lab');
    fileID = fopen(filename,'w');
    fprintf(fileID,'%d %d %d\n',f');
    fclose(fileID);
end
```

```
function[membership_idx] = myKmeans(meanstore_tot,thr)

clear all
close all
clc

% load data
m = load ('/Users/negarolfati/Downloads/voicebox/meanstore_tot.mat');
m = m.meanstore_tot ;


N = size(m,2);  % number of data points
numclust =2;

% Find the main centroid

main_centr = mean(m,2);

initGlobalDist = 100e100;

%% Initialise centroids

centers_init = cell(1,numclust);

for k =1 : numclust/2
    centers_init{1,2*k-1} = main_centr;
    centers_init{1,2*k} = main_centr + 0.1*randn(22,1); % datadim = 2
end
```

```
all_global_dist = [initGlobalDist];
% Create an outer loop to split the clusters

l = 1; % outer loop counter

while 1
%for l = 1:max_iter
   'new cluster set'

   if(numclust > N)
     % if (2^max_iter > N)
      error('Error! Number of clausters more than number of datapoints');
   end

   membership_idx = zeros(N,1);
   dist = zeros(numclust, N);
   centers = centers_init;
   temp_idx = zeros (1,N);

   %figure
   flag = 1;
   % Create an innerloop to stabilize the centroids.
   while flag

      %**** 1.partition
      for k = 1:numclust
         c = centers{1,k};
         dist(k,:) = sqrt(sum((m-repmat(c,1,N)).^2));

      end

      [sorted, ind] = sort(dist);
      membership_idx = ind(1,:);


      % compute the global distortion
      localDist = zeros(1,numclust);

      for k = 1:numclust
         a = m(:,(membership_idx==k));
         M = size(a,2);
         c = centers{1,k};
         localDist(1,k) = sum(sqrt(sum((a-repmat(c,1,M)).^2)));  %Distortion inside each cluster
      end

      % Compute the distortion reduction compared to previous step
      globalDist = sum(localDist(:));
      G = (initGlobalDist-globalDist)/initGlobalDist;

      if (all(membership_idx == temp_idx)  | (G < 1e-3))
         flag = 0;
         'breaking'
         all_global_dist = [all_global_dist globalDist];
         break
      else
         temp_idx = membership_idx;
         initGlobalDist = globalDist;
      end
```

```
        %**** 2. Centroid update

        for k = 1:numclust
            a = m(:,(membership_idx==k));
            if size(a,1)==1
                centers{1,k} = a;
            else
                centers{1,k} = mean(a,2);
            end
        end

    end


    %%
    % ****** How far split the clusters? We need a stop criterion!


    iter_cond = abs(all_global_dist(l) - all_global_dist(l+1)) / all_global_dist(l)

    if (iter_cond < 0.104)
        break;
    end


    %**** 3.split the clusters into two, and prepare data for next iteration

    centers_old = centers;
    numclustnew = 2*numclust;
    numclust = numclustnew;

    centers_init = cell(1 ,numclustnew);


    for k = 1:numclust/2

        centers_init{1,2*k-1} = centers{1,k};
        centers_init{1,2*k} = centers{1,k} + 0.1*randn(22,1); % 22 = datadim

    end
    l = l+1;

end
```

APPENDIX C: K-MEANS ALGORITHM FOR CLUSTERING TWO-DIMENSIONAL DATA INTO UN-PREDEFINED NUMBER OF CLUSTERS.

```
clear all
close all
clc

m = [-0.06 0.9;0.3 -0.5;-0.4 -0.1;-0.1 -0.8;0.01 0.4;2.9 2.0;2.2 1.8;1.8 1.1;
   1.8 2.1;1.8 1.8;-1.3 -2.6;-2.1 -2.5;-2.4 -2.4;-1.8 -2.1;-2.0 -2.8;
   3.4 4.9;3.4 5.3;3.5  4.8;3.3 4.6;3.3 5.1];
m = m';

N = size(m,2);  % number of data points
numclust =2;  % the whole data space to be splitted into 2 clusters.

% Find the main centroid
figure;
main_centr = mean(m,2);
scatter(m(1,:),m(2,:),'b','filled');
hold on

scatter(main_centr(1,:),main_centr(2,:),'r','*','SizeData' ,100);
pause
% Define initial distortion
initGlobalDist = 10e10;
hold off


%% Initialise centroids

centers_init = cell(1,numclust);

for k =1 : numclust/2
   centers_init{1,2*k-1} = main_centr;
   centers_init{1,2*k} = main_centr + 0.1*randn(2,1); % datadim = 2
end

all_global_dist = [initGlobalDist];
% Create an outeter loop to control splitting the clusters
l = 1; % outer loop counter
while 1
   'new cluster set'
   if(numclust > N)
      %if (2^max_iter > N)
      error('Error! Number of clausters more than number of datapoints');
   end

   membership_idx = zeros(N,1);
   dist = zeros(numclust, N);
   centers = centers_init;
   temp_idx = zeros (1,N);

   figure

   scatter(m(1,:),m(2,:),'b','filled');
   hold on
   for  k = 1:numclust
      c = centers{1,k};
```

```
    scatter(c(1,:),c(2,:),'r','*','SizeData' ,100)
  end
  title('The splitted centroids')
  pause
  hold off

  flag = 1;
  % Create an inner loop to stabilize the centroids.
  while flag
    figure
    %**** 1. Partition the data points by assigning them to their nearest centroid.
    for k = 1:numclust
      c = centers{1,k};
      dist(k,:) = sqrt(sum((m-repmat(c,1,N)).^2));

    end

    [sorted, ind] = sort(dist);
    membership_idx = ind(1,:);


    % compute the global distortion
    localDist = zeros(1,numclust);

    for k = 1:numclust
      a = m(:,(membership_idx==k));
      M = size(a,2);
      c = centers{1,k};
      localDist(1,k) = sum(sqrt(sum((a-repmat(c,1,M)).^2)));  %Distortion inside each cluster
    end

    % Compute the global distortion reduction compared to previous step
    globalDist = sum(localDist(:));
    G = (initGlobalDist-globalDist)/initGlobalDist

    if (all(membership_idx == temp_idx)  | (G < 1e-5))
      flag = 0;
      'breaking'
      all_global_dist = [all_global_dist globalDist];
      break
    else
      temp_idx = membership_idx; % copy the membership index and the local distortion into temporary
variables
      initGlobalDist = globalDist;
    end

    %**** 2. Centroid Update

    scatter(m(1,:),m(2,:),'b','filled');

    hold on
    for k = 1:numclust
      a = m(:,(membership_idx==k));
      if size(a,1)==1
        centers{1,k} = a;
      else
        centers{1,k} = mean(a,2);
      end
    end
```

```
        for k = 1:numclust
            c = centers{1,k};
            scatter(c(1,:),c(2,:),'r','*','SizeData' ,100)
        end

        title('The updated centroids')
        pause

        hold off
    end


    %% How far split the clusters? We need to define a stop criterion.


    iter_cond = abs(all_global_dist(l) - all_global_dist(l+1)) / all_global_dist(l);

    if (iter_cond < 0.8)
        break;
    end


    %**** 3.split the centroid into two and prepare data for next iteration

    centers_old = centers;
    numclustnew = 2*numclust;
    numclust = numclustnew;

    centers_init = cell(1 ,numclustnew);


    for k = 1:numclust/2

        centers_init{1,2*k-1} = centers{1,k};
        centers_init{1,2*k} = centers{1,k} + 0.1*randn(2,1); % 22 = datadim

    end
    l = l+1;

end
```

APPENDIX D: THE FIRST EXPERIMENT: RECOGNITION

Script: ~/Detection/AF_TIMIT/1_state/trainHMMs/firstApproach.sh

```
#!/bin/bash

TOP=`pwd`

# Create 6 folders in your main directory.

for i in 0 1 2 3 4 5
do
mkdir -p hmm$i
done


# Create the folder LOG to store all logs.

mkdir -p LOG
# Delete labellist if it exists (Is there a better way to prevent multiple-writing on it?)

rm -f labellist
# Make a list of symbols in the file labellist

for i in {1..128}
do
echo "L$i" >> labellist
done

#Create dictionary
rm -f dict
for i in {1..128}
do
echo "L$i L$i" >> dict
done


# Create train.scp : a script file containing a list of all feature filenames (*.fea). Copy the existing list, and edit paths and extensions.

cp ../extract_melbankenergies/data/train.scp train.scp
sed -i -e 's;^; features/;' train.scp
sed -i '.wav' 's/\.wav/\.fea/' train.scp

# Repeat the same to create test.scp

cp ../extract_melbankenergies/data/test.scp test.scp
sed -i -e 's;^; features/;' test.scp
sed -i '.wav' 's/\.wav/\.fea/' test.scp


# Initialise proto

HCompV -A -D -T 1 -C config -f 0.01 -m -S train.scp -M hmm0 proto> LOG/HCompV.log
```

```
# Make a copy of proto content 512 times into hmm0/hmmdefs

(perl -0777pe '$_=$_ x 128' hmm0/proto )> hmm0/hmmdefs

# Remove the undesired lines from hmmdefs

sed -i " "/<STREAMINFO> 1 23/d" hmm0/hmmdefs
sed -i " "/<VECSIZE> 23<NULLD><FBANK><DIAGC>/d" hmm0/hmmdefs
sed -i " "/~o/d" hmm0/hmmdefs


# edit the strings "proto" in hmmdefs according to the labels exisiting in labellist. (Without copying labellist into hmm0, hmmdefs becomes empty after running awk.(why?))
cp labellist hmm0/labellist
cd hmm0
awk 'FNR==NR{a[++i]=$0; next} /proto/{sub(/proto/, a[++j])} 1' labellist hmmdefs >> tmp && mv tmp hmmdefs


# Remember to create macros MANUALLY in hmm0


# Add L to all label files corresponding to training and testing data, and store the files into LABELSTRAIN and LABELSTEST
#(output from script Centroidfinding.m are in form of string of numbers. add L before them)

cd $TOP
mkdir -p LABELSTRAIN
mkdir -p LABELSTEST
mkdir -p Results
cd ../mergedlabels_train
for i in *;
do
sed 's/^/L/' "$i" > ../trainHMMs/LABELSTRAIN/$i
done

cd ../mergedlabels_test
for i in *;
do
sed 's/^/L/' "$i" > ../trainHMMs/LABELSTEST/$i
done


# Go back to main directory

cd $TOP

# Create MLFs from training and testing label files

HLEd -l '*' -i labels_train.mlf dummy.led LABELSTRAIN/*
HLEd -l '*' -i labels_test.mlf dummy.led LABELSTEST/*


#Create Grammar.net from the command-line

# Create lattice. This to be used in decoding
Hparse Grammar.net Grammar.lat
```

Appendix D

```
# Now train, decode and evaluate iteratively

for((i=0;$i<5;i++));
do

echo Iteration number $((i+1))

HERest -A -D -T 1 -C config -I labels_train.mlf -t 150.0 100.0 500.0 -S train.scp -H hmm$i/macros -H
hmm$i/hmmdefs -M hmm$((i+1)) labellist> LOG/Rec./HERest30$((i+1)).log
echo Training finished!

HVite -A -D -C config -w Grammar.lat -H hmm$((i+1))/hmmdefs -H hmm$((i+1))/macros -S test.scp -i
results$((i+1)).mlf  -p -30.0 dict labellist > LOG/Rec./HVite30$((i+1)).log
echo Decoding finished!

HResults -A -D -I labels_test.mlf labellist results$((i+1)).mlf >& Results/Rec./HResults30$((i+1)).txt

echo Results stored in Results/Rec./HResults30$((i+1)).txt
done
```

APPENDIX E: THE SECOND EXPERIMENT: FORCED ALIGNMENT

Script: ~/Detection/AF_TIMIT/1_state/trainHMMs/secondApproach.sh

```
#!/bin/bash

TOP=`pwd`

rm -f labellist
# Make a list of symbols in the file labellist

for i in {1..128}
do
echo "L$i" >> labellist
done

mkdir -p TimedLABELSTRAIN
mkdir -p TimedLABELSTEST


cd ../timedlabels_train
for i in *;
do
sed  -E 's/(.* .*) /\1 L/' "$i" > ../trainHMMs/TimedLABELSTRAIN/$i
done

cd ../timedlabels_test
for i in *;
do
sed  -E 's/(.* .*) /\1 L/' "$i" > ../trainHMMs/TimedLABELSTEST/$i
done

cd $TOP

for i in {1..6}
do
mkdir -p dir$i
done
mkdir -p Results

for i in {1..6}
do
rm -rf dir$i/*
done

HLEd -l '*' -i timedlabels_train.mlf dummy.led TimedLABELSTRAIN/*
HLEd -l '*' -i timedlabels_test.mlf dummy.led TimedLABELSTEST/*


# Initialise HMM models individually
initprotos.pl labellist

HHEd -w N1.mmf -d dir1 dummy.led labellist

for i in {1..5}
```

```
do


# Perform forced alignment
HVite -A -D -T 1 -l '*' -o M  -C config -H N$i.mmf -i aligned$i.mlf  -m  -y lab -a -I timedlabels_train.mlf -S
train.scp  -p -30.0 dict labellist> LOG/F.A./HVite_aligned_train$i
echo the aligned mlfs stored in Aligned$i.mlf
#Train using HRest
trainafteralignment.pl $i


HHEd -w N$((i+1)).mmf -d dir$((i+1)) dummy.led labellist
# Decode
HVite -A -D -C config -w Grammar.lat -H N$((i+1)).mmf -S test.scp -i results$i.mlf  -p -30.0 dict labellist >
LOG/F.A./HVite_Decode_30$i.log
# Evaluate
HResults -A -D -I timedlabels_test.mlf labellist results$i.mlf >& Results/F.A./HResults$i.txt
echo Results stored in Results/F.A./HResults$i.txt
done
```

Script: ~/Detection/AF_TIMIT/1_state/trainHMMs/initprotos.pl

```
#!/usr/bin/perl

# usage: initprotos.pl labellist
# labellist: text file containing model names (L1,L2,..,L512)

$model = $ARGV[0];

open(CF, "< $model") || die "can't open $model: $!";
my @segmentmodels=();
while(<CF>){
   chomp;
   push(@segmentmodels,$_);
}
close(CF);

foreach $model (@segmentmodels)
{
$cmd = "HInit -I timedlabels_train.mlf -i 3 -l $model -o $model -S train.scp -M dir1 proto";
   print $cmd, "\n";
   #next;
system("$cmd");
}
```

Script: ~/Detection/AF_TIMIT/1_state/trainHMMs/trainafteralignment.pl

```
#!/usr/bin/perl
# usage: trainafteralignment.pl i


use strict;
use warnings;
```

```
use File::Spec::Functions qw/ catdir catfile /;

my $iter = $ARGV[0];
my $dir =  'dir'.$iter;
my $dir2 = 'dir'.($iter+1);
my $alignedMLF = 'aligned'.$iter.'.mlf';
opendir my ($dh), $dir;
while (my $node = readdir $dh) {
    my $file = catfile($dir, $node);
    next unless -f $file;
    my $cmd = "HRest -T 1 -I $alignedMLF -i 20 -l '$node' -M '$dir2' -S train.scp '$file'";
     #print $cmd, "\n";
    #next;
    system($cmd);
}
closedir $dh;
```