On Developing Acoustic Models Using HTK

M.A. Spaans BSc.

On Developing Acoustic Models Using HTK

M.A. Spaans BSc.

Delft, December 2004

Delft University of Technology

December, 2004.

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands
E-mail: mike@ch.tudelft.nl

Research was done at the Asahi Kasei Information Technology Laboratory
Asahi Kasei Corporation
22F Atsugi AXT Main Tower
3050 Okada, Atsugi, Kanagawa
Japan

This Master's Thesis was submitted to the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology in partial fulfillment of the requirements for the degree of *Master of Science*.

Members of the Committee:
drs.dr. L.J.M. Rothkrantz
dr. K. van der Meer
dr. C.J. Veenman
ir. P. Wiggers

"Aut viam inveniam aut faciam."

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The interaction between humans and technology is changing and automatic speech recognition is a driving force in this process. Speech recognition technology is changing the way information is accessed, tasks are accomplished and business is done. The growth of speech recognition applications over the past years has been remarkable. From high-priced, limited dictation systems to affordable products capable of understanding natural speech, operating both at home and in professional environments. An important factor in this growth is the mobile telecommunications industry, which provides demand for speech recognition systems and also drives the development of signal processing technology, essential to speech recognition.

In modern society people typically interact with several electronic devices during a day, ranging from mobile phones and personal digital assistants to photocopiers and common household appliances. The machine, however, dictates the user interaction and requires the user to adapt to its unnatural, and often complex ways. Spoken language technology will enable access to machines to become faster and easier. It is the most natural interface method possible.

Automatic speech recognition research is several decades old. In the 1970s significant technological breakthroughs were made relating to the modeling of human speech sounds using hidden Markov models. Hidden Markov models are still the cornerstone of contemporary speech recognition systems. In this period, speech recognition research was performed mainly in universities and government-funded programs. Since the 1990s private companies have had an active interest in speech recognition. An overview of the history of speech recognition research can be found in appendix A.

1.1 The Speech Recognition Problem

Automatic speech recognition is essentially the process of mapping an acoustic signal, captured by a microphone, telephone or other acoustical transducer, to a sequence of discrete entities, such as phonemes or words. A typical speech recognition system consists of several components, as is illustrated in figure 1.1. In this figure, the *decoder* provides the external *application* with recognition results in order for it to perform required actions. The *acoustic models* include the representation of knowledge about acoustics, phonetics, signal variability,

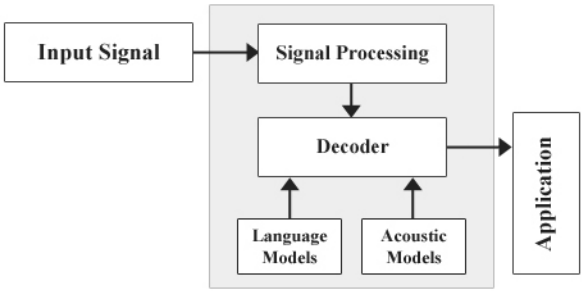

Figure 1.1 A typical speech recognition system.

etc. The *language models* refer to a system's knowledge of which words it is able to recognize, which words are likely to occur together, and in what order. The voice *input signal* is processed by the *signal processing* unit, which extracts feature vectors from the input signal for the decoder. The decoder uses the acoustic and the language models to generate the word sequence with the maximum probability given the input feature vectors.

Automatic speech recognition systems can be characterized by many parameters [5], some of which are shown in table 1.1. In an isolated-word speech

Table 1.1 Parameters that characterize speech recognition systems.

Parameter	Range
Speaking Mode	Isolated words to continuous speech
Speaking Style	Read speech to spontaneous speech
Enrollment	Speaker-dependent to Speaker-independent
Vocabulary	Small (less than 20 words) to large (more than 20,000 words)
Language Model	Finite-state to context-sensitive
SNR	High (more than 30 dB) to low (less than 10 dB)
Transducer	Noise-canceling microphone to telephone

recognition system, the user is required to pause briefly between words, which is not required in a continuous speech recognition system. Systems can be designed to handle spontaneous speech, which is much more difficult to recognize than speech spoken from a written source. In some systems enrollment is required, which means a user has to provide the system with several speech samples before being able to use it properly. This is not required in speaker-independent systems.

Some parameters are related to the task of the system. When using a large vocabulary, recognition will be more difficult than if a small vocabulary is used. If system input is a sequence of words, language models are used to restrict the number of allowed combinations. A finite-state model is a very simple network specifying the allowable order of words explicitly. Context-sensitive language models are more complex and are used to approximate natural spoken language. A final set of parameters include properties of the environmental noise and the type and placement of the microphone.

Recognition of human speech is considered a difficult problem, mainly due to two factors [16]. First, the acoustic realization of phonemes is highly depen-

dent on the context in which they appear, making it hard to explicitly identify their boundaries. This problem is the result of coarticulation, which is the blending of the articulation of a sound into the articulation of the following and preceding sounds. To avoid this problem it is possible to restrict the underlying recognizable entities to words instead of phonemes. Though far less powerful, word-based models nevertheless have a wide range of practical applications. Coarticulation and the production of human speech sounds is discussed in chapter 2.

The second factor is the large variability in characteristics of the speech signal. This variability has three main components: linguistic variability, speaker variability and channel variability. Linguistic variability refers to effects of phonetics, syntax, etc. Speaker variability includes intra- and interspeaker variability and results from differences in speaking rate, voice quality, etc. Channel variability include the effects of background noise and properties of the transmission channel. Variability and robustness in speech recognition is discussed in chapter 4.

1.2 Speech Recognition at Asahi Kasei

This section will focus on speech recognition at Asahi Kasei. The Asahi Kasei Corporation of Japan is a holding company for seven subsidiary business units in diverse areas. These include Asahi Kasei Fibers, Asahi Kasei Life & Living and Asahi Kasei Chemicals. The holding company provides common services for all subsidiaries and also controls corporate Research & Development (R&D). Corporate R&D is divided into four laboratories:

- Central Research Laboratory. Research is focussed on biotechnology and nanotechnology.
- Membrane Technology Laboratory. Research is focussed on membrane technology.
- Analysis & Simulation Center. Focus is on analysis and computer simulation technology.
- Information Technology Laboratory. Development of innovative software solutions based on pattern recognition and digital signal processing technology.

In August 2000 the Voice Interface Project was launched at the Asahi Kasei Information Technology Laboratory, combining several areas of research related to speech technology from the Central Research Laboratory. The aim of the Voice Interface Project is to provide commercially viable speech recognition, voice compression and text-to-speech middleware solutions. The Voice Interface Project's flagship product is VORERO (Voice Recognition Robust), voice recognition middleware. Other products include voice compression/decompression middleware (MMEV), Japanese text-to-speech middleware (VOStalk) and hands-free middleware (VOCLE).

1.2.1 VORERO

VORERO is essentially a voice recognition middleware platform. Its primary target is embedded systems and it is currently employed in car navigation systems, cellular phones, personal digital assistants and robotics. VORERO is based on hidden Markov models and allows phoneme and word models to be used simultaneously. It includes advanced acoustic analysis, which provides robustness through noise reduction and echo cancellation techniques. VORERO speech recognition is speaker independent and uses a small amount of system memory. The general VORERO system architecture is similar to the typical speech recognition system described in the previous section and is illustrated in figure 1.2. The VORERO *engine* provides the speech recognition using VORERO *data*, which consists of a set of acoustic models, a dictionary and a vocabulary network. The network specifies the recognition task to be performed (i.e. what words can be recognized and in what order). Consumer application can interface the VORERO through the Software Development Kit (SDK), which is a set of libraries that encapsulate the VORERO engine.

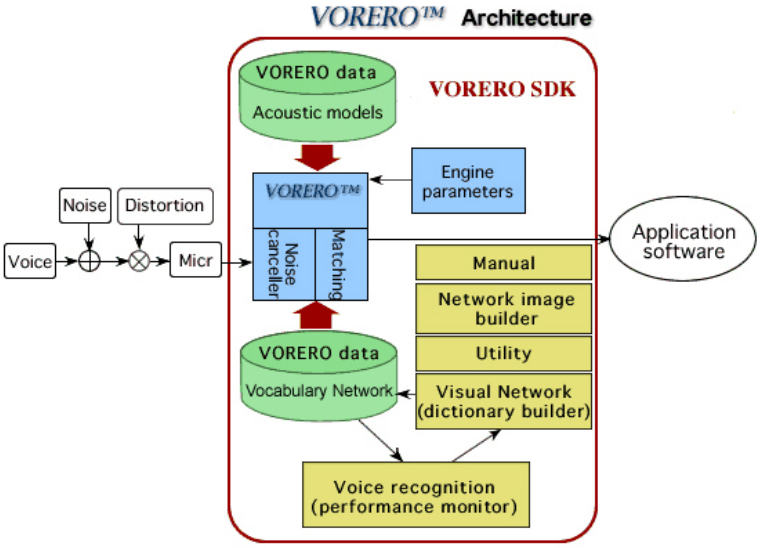

Figure 1.2 VORERO architecture.

1.3 Research Objectives

VORERO supports a number of languages, including Japanese, North American English, Korean, Mandarin, German, Spanish and French. For the VORERO SDK release 6.0, additional support for the languages Italian, Dutch and Portuguese was desired. The research objective described in this thesis is *the addition of support for the Dutch language to VORERO*. The followings tasks contribute to achieving the research objective:

- Understanding of current speech recognition technology, by studying relevant literature.
- Understanding of the mathematical principles involved in stochastic speech recognition using hidden Markov model theory.
- Study of the Dutch phoneme set and Dutch pronunciation rules.
- Design of the Dutch acoustic models using the Hidden Markov Toolkit (HTK).
- Training of the Dutch acoustic models using the HTK.
- Evaluation and optimization of Dutch speech recognition.
- Design of the Dutch pronunciation dictionary.
- Addition of the Dutch language to the VORERO SDK 6.0.

1.3.1 About this Document

This document represents the partial fulfillment of the requirements for obtaining a Master of Science degree. Each of the research tasks listed above will be discussed in detail. In chapter 2 the production of human speech is analyzed, the concept of a phoneme is introduced and the complete Dutch phoneme set is described. Chapter 3 is a thorough review of all the mathematical principles involved in automatic speech recognition. Special focus is on hidden Markov model theory. Chapter 4 is the result of studying relevant literature related to contemporary speech recognition. In this chapter a number of key challenges will be discussed. These challenges were identified in a survey on Human Language Technology, funded by the U.S. National Science Foundation.

In chapter 5 and 6 the design of acoustic models will be discussed. Chapter 5 describes the Hidden Markov Toolkit (HTK) and its training tools. Chapter 6 represents the core of the training of the Dutch acoustic models. All required steps from the preparation of data to the evaluation of the models are described in detail. Chapter 7 introduces a number of techniques related to optimizing the Dutch acoustic models. In the final chapter the conclusion and final remarks will be presented.

Chapter 2

Human Speech

In this chapter the basics of human speech production are discussed. The concept of a *phoneme* is introduced, and the complete phoneme set of the Dutch language is described in detail. This chapter also covers differences in phoneme realization depending on context.

2.1 Human Speech Systems

In this section, the organs responsible for the production of speech will be described. Essentially, the speech organs can be divided into two groups, each with a specific function in the production of speech. These groups are the *subglottal system* and the *supraglottal system* and are illustrated in figure 2.1. The names subglottal and supraglottal refer to the position relative to the *glottis*, the space between the vocal folds.

2.1.1 Subglottal System

The main function of the subglottal system is to provide the body with oxygen, by inhalation of air. The exhalation of air provides the source of energy for speech production. The subglottal systems consists of the *lungs*, the *trachea* (or windpipe) and the *larynx* (or voicebox). The larynx sits at the top of the trachea and contains the vocal folds, essential in speech production. The vocal folds are two bands of ligament and muscle that vibrate during the articulation of vowels and various consonants. The space between them is called the glottis. Its shape determines the production of *voiced* or *unvoiced* sounds, which will be described in the next section. The length of the vocal folds depends on age and gender: women have significantly shorter vocal folds than men and the vocal folds of children are smaller still.

2.1.2 Supraglottal System

The supraglottal system consists of the *pharynx* (or throat) and the articulators in the oral cavity. The position and movement of the articulators determine the properties of produced speech sounds. The articulators in the oral cavity are:

- Lips (or labia).

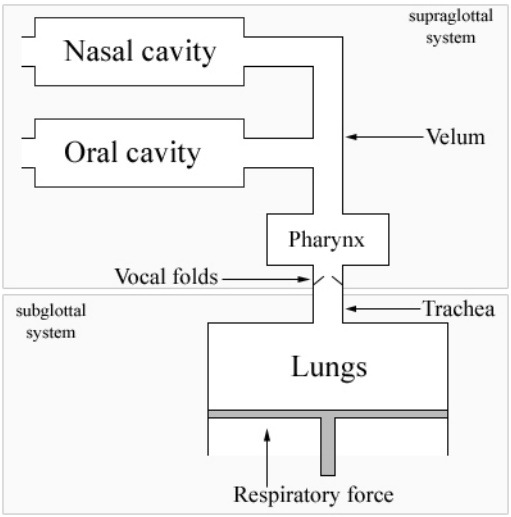

Figure 2.1 Diagram of human speech organs.

- Teeth (or dentes).
- The alveolar ridge, a small protuberance behind the upper teeth.
- The palate (or palatum), an arched bony structure that forms the forward part of the roof of the mouth.
- The velum, part of the roof of the mouth, which can be lowered to allow the passage of air through the nasal cavity or raised to block the nasal cavity. When the nasal cavity is opened nasalized sounds are produced.
- The uvula, the soft, fleshy tip of the velum.
- The tongue, which basically covers the whole floor of the oral cavity. Five regions can be distinguished: the tip (or apex), the blade (or lamina), the front (or antero-dorsum), the back (or dorsum) and the root (or radix).

2.2 Speech Production

The production of speech involves three phases. In the *initiation phase* a flow of air is set in motion by an initiator, such as the lungs. The flow of air is turned into a pulsating stream in the *phonation phase*. This is caused by repeated opening and closing of the glottis: the vibration of the vocal folds. This phase is skipped in unvoiced speech sounds. The final phase is the *articulation phase*, in which vowels and consonants are formed by the application of the various articulators, resulting in resonance of the pulsating airstream in the pharynx, the oral cavity and/or the nasal cavity.

In human speech science, a basic unit of speech is referred to as a *phoneme*. A phoneme is esentially a set of all possible ways of pronouncing a specific speech sound. An actual realization of a phoneme is called an *allophone* (or

phone). Phonemes are typically categorized as vowels or consonants. During articulation of consonants, the flow of air is in some way obstructed in the vocal tract and the vocal folds may or may not vibrate. The flow of air is always free during articulation of vowels and the vocal folds always vibrate.

The rest of this section will cover the definition of the phoneme set of the Dutch language in detail. For the notation of the phonemes SAMPA symbols will be used.

2.2.1 Consonants

Consonants can be classified on the basis of their manner and place of articulation, and the state of the vocal folds. The manner of articulation refers to the degree and type of constriction in the vocal tract. Consonants can be either *voiced* or *unvoiced*, depending on the state of the vocal folds. If the vocal folds vibrate during speech production, the consonant is voiced. If the vocal folds do not vibrate, the consonant is unvoiced.

The consonants can be divided between the *plosive* and *fricative* consonants, known as the *obstruents* (or real consonants), and the remaining consonants, known as the *sonorants*. Sonorant consonants include the *nasals* and the *approximants*. The approximant consonants can be further divided into *liquids* and *glides*. This division is illustrated in figure 2.2.

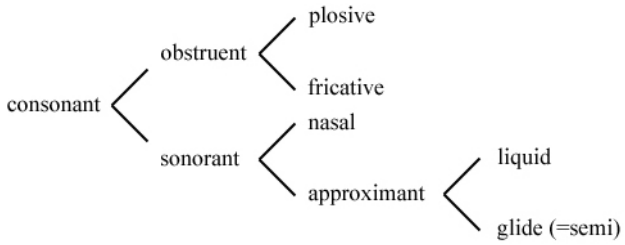

Figure 2.2 Dutch consonant classification.

Plosives

A plosive consonant is produced by a complete obstruction of the air flow in the vocal tract. The pressure builds up behind the obstruction, which causes the air to rush out with an 'explosive' sound when released. Table 2.1 lists the six plosive consonants found in the Dutch language and their place of articulation.

Table 2.1 Plosives

	labial/ labiodental	**alveolar**	**postalveolar/ palatal**	**velar/uvular/ glottal**
voiced	/b/ *b*ak	/d/ *d*ak		/g/ *g*oal
unvoiced	/p/ *p*ak	/t/ *t*ak		/k/ *k*at

Fricatives

Fricative consonants are produced by air being pushed through a constriction in the vocal tract. If enough air is pushed through the constriction, an area of turbulence will be formed, which will be perceived as noise. The constriction is formed by close approximation of two articulators. Table 2.2 lists the eight fricative consonants found in the Dutch language and their place of articulation.

Table 2.2 Fricatives

	labial/ labiodental	**alveolar**	**postalveolar/ palatal**	**velar/uvular/ glottal**
voiced	/v/ *v*el	/z/ *z*ak /Z/ baga*g*e		/G/ *g*oed
unvoiced	/f/ *f*el	/s/ *s*ok /S/ *sj*aal		/x/ to*ch*

Nasals

Nasals are produced by the flow of air moving through the nasal cavity, which is accessible by the lowering of the velum. The oral cavity is obstructed by an articulator. All nasals are voiced. Table 2.3 lists the three nasal sounds found in the Dutch language and their place of articulation.

Table 2.3 Nasals

labial/ labiodental	**alveolar**	**postalveolar/ palatal**	**velar/uvular/ glottal**
/m/ *m*an	/n/ *n*on		/N/ ba*ng*

Approximants

Approximants are formed by two articulators approaching each other but not close enough for an area of turbulence to be formed. The Dutch liquids are /l/ and /r/. The /r/ is a little bit complicated as it has a number of possible realizations, that differ in the place of articulation, the number of contact moments of the articulator and whether the articulation is continuous or not. Glides are /w/ and /j/. The /h/ phoneme can be considered both an approximant and a glottal fricative. Table 2.4 lists the approximants found in the Dutch language and their place of articulation.

Table 2.4 Approximants

labial/ labiodental	**alveolar**	**postalveolar/ palatal**	**velar/uvular/ glottal**
/w/ *w*eer	/r/ *r*and /l/ *l*am	/j/ *j*as	/h/ *h*oed /r/ pee*r*

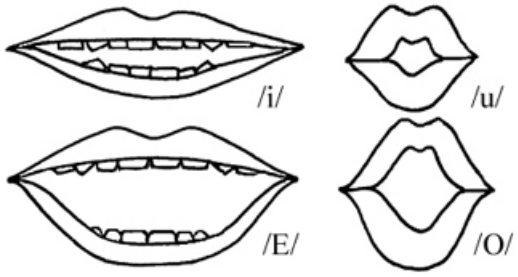

Figure 2.3 Dutch vowel articulation.

2.2.2 Vowels

Vowels differ from consonants in the fact that the air flow from the lungs is not constricted in the oral cavity. Vowels can be classified on the basis of tongue position and rounding of the lips. The tongue plays a major role in the production of vowels. Its movement determines the manner in which the flow of air resonates in the pharynx and oral cavity. The tongue position can be analyzed in the vertical dimension (high, middle, low) and the horizontal dimension (front, central, back). The lips further determine vowel articulation and can be either rounded or spread as illustrated in figure 2.3. Table 2.5 lists the vowels found in the Dutch language and the manner of articulation. It is also possible to characterize vowels by the amount of tension of the speech muscles, which can be either 'tense' or 'lax', though this is considered a minor phonological feature.

Table 2.5 Dutch vowels.

		spread	**rounded**	
		front	**central**	**back**
high	**tense**	/i/ p*ie*t	/y/ f*uu*t	/u/ v*oe*t
middle	**lax**	/I/ p*i*t	/Y/ p*u*t	/O/ p*o*t
	tense	/e:/ v*ee*l	/2:/ b*eu*k	/o:/ b*oo*t
low	**diphthong**	/Ei/ st*ij*l	/9y/ h*ui*s	/Au/ r*ou*w
	lax	/E/ p*e*t		/A/ p*a*t
	tense			/a:/ p*aa*l

A special class of vowels are *diphthonic* vowels (diphthongs). To produce diphthongs, the articulator are moved from one to another configuration. The diphthongs listed in table 2.5 are referred to as 'real' diphthongs, while table 2.6 lists the 'possible' diphthongs in the Dutch language.

A final class of Dutch vowels are the *marginal* vowels. These are not native to the Dutch language and are mainly found in loan words. They are listed in table 2.7

Table 2.6 Dutch diphthonic vowels.

SAMPA symbol	example word
/a:i/	dr*aai*
/o:i/	m*ooi*
/ui/	r*oei*
/iu/	n*ieuw*
/yu/	d*uw*
/e:u/	sn*eeuw*

Table 2.7 Dutch marginal vowels.

SAMPA symbol	example word
/i:/	anal*y*se
/y:/	centrif*u*ge
/u:/	cr*ui*se
/E:/	crème
/9:/	fr*eu*le
/O:/	z*o*ne
/A:/	b*a*sketbal
/Y~/	parf*um*
/E~/	bullet*in*
/O~/	chans*on*
/A~/	g*en*re

2.3 Sound and Context

The position of the articulators involved in producing speech sounds does not change abruptly from one speech segment to another. This transition gradual and fluid, which leads to an effect called *coarticulation*. Coarticulation is literally the process of joint articulation, which means that different speech sounds are articulated simultaneously. If a speech sound is influenced by sounds that are still unspoken, the coarticulatory effect is referred to as *anticipation*. If a speech sound is still not fully realized due to the previous sounds, the coarticulatory effect is referred to as *perseverance*.

As was mentioned in the previous section, an actual realization of a phoneme is referred to as an *allophone*. Allophonic realizations of phoneme differ between speakers and even a single speaker will never really produce exactly the same speech sounds. Allophonic realization of phonemes, however, also depend heavily on the context in which they are produced. If different sounds precede and follow a particular phoneme, its realization will be affected. An example is the /l/ sound in the words 'like' and 'kill'.

In general it is harder to become aware of coarticulatory effects than of allophonic alternatives, though both form a serious obstacle in automatic speech recognition.

Chapter 3

Speech Recognition

In this chapter some of the essential mathematical theory related to speech recognition will be discussed. This includes *hidden Markov model* theory, cornerstone of contemporary speech recognition systems.

3.1 System Overview

The goal of speech recognition can be formulated as follows: for a given acoustic observation $\mathbf{X} = X_1, X_2, \ldots, X_n$, find the corresponding sequence of words $\hat{\mathbf{W}} = w_1, w_2, \ldots, w_m$ with maximum *a posteriori* probability $P(\mathbf{W}|\mathbf{X})$. Using *Bayes' decision rule*, this can be expressed as:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{X}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{X})} \tag{3.1}$$

Since the acoustic observation $\mathbf{X}$ is fixed, equation 3.1 is equal to:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{X}|\mathbf{W})P(\mathbf{W}) \tag{3.2}$$

Probability $P(\mathbf{W})$ is the *a priori* probability of observing $\mathbf{W}$ independent of the acoustic observation and is referred to as a *language model*. Probability $P(\mathbf{X}|\mathbf{W})$ is the probability of observing acoustic observation $\mathbf{X}$ given a specific word sequence $\mathbf{W}$ and is determined by an *acoustic model*. In pattern recognition theory, the probability $P(\mathbf{X}|\mathbf{W})$ is referred to as the *likelihood* function. It measures how likely it is that the underlying parametric model of $\mathbf{W}$ will generate observation $\mathbf{X}$.

In a typical speech recognition process, a word sequence $\mathbf{W}$ is postulated and its probability determined by the language model. Each word is then converted into a sequence of phonemes using a pronunciation dictionary, also known as the *lexicon*. For each phoneme there is a corresponding statistical model called a *hidden Markov model* (HMM). The sequence of HMMs needed to represent the utterance are concatenated to a single composite model and the probability $P(\mathbf{X}|\mathbf{W})$ of this model generating observation $\mathbf{X}$ is calculated. This process is repeated for all word sequences and the most likely sequence is selected as the recognizer output.

Most contemporary speech recognition systems share an architecture as illustrated in figure 3.1. The acoustic observations are represented by *feature*

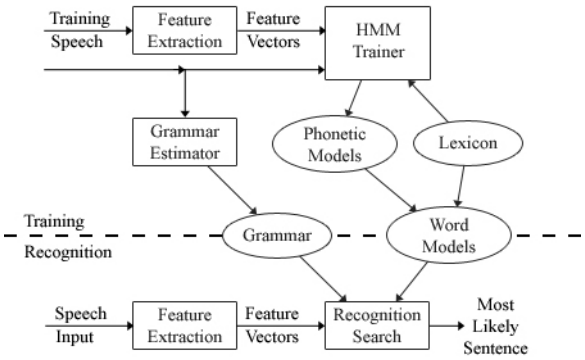

Figure 3.1 A general system for training and recognition.

vectors. Choosing appropriate feature vectors is essential to good speech recognition. The process of extracting features from speech waveforms will be described in detail in the next section. Hidden Markov models are used almost exclusively for acoustic modeling in modern speech recognition systems. Hidden Markov model theory is described in detail in section 3.3 and the application of HMMs to acoustic modeling in section 3.4. Section 3.5 focusses on language modeling and section 3.6 on the speech recognition decoding process.

3.2 Acoustic Analysis

The acoustic analysis is the process of extracting feature vectors from input speech signals (i.e. waveforms). A feature vector is essentially a parametric representation of a speech signal, containing the most important information and stored in a compact way. In most speech recognition systems, some form of preprocessing is applied to the speech signal (i.e. applying transformations and filters), to reduce noise and correlation and extract a good set of feature vectors. In figure 3.2 the process of extracting feature vectors is illustrated. The speech signal is divided into analysis frames at a certain *frame rate*. The size of these frames is often 10 ms, the period that speech is assumed to be stationary for. Features are extracted from an analysis window. The size of this window is independent of the frame rate. Usually the window size is larger than the frame rate, leading to successive windows overlapping, as is illustrated in figure 3.2.

Much work is done in the field of signal processing and several methods of speech analysis exist. Two of the most popular will be discussed: *linear predictive coding* and *Mel-frequency cepstral analysis*.

3.2.1 Linear Predictive Coding

Linear predictive coding (LPC) is a fast, simple and effective way of estimating the main parameters of speech. In linear predictive coding the human vocal tract is modeled as an *infinite impulse response* filter system that produces the speech signal. This modeling produces an accurate representation of vowel sounds and other voice speech segments that have a resonant structure and a

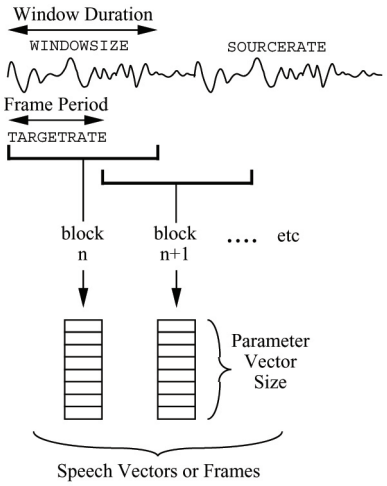

Figure 3.2 Feature extraction.

high degree of similarity over time shifts that are multiples of their pitch period. The linear prediction problem can be stated as finding the coefficients a_k, which result in the best prediction (that minimizes the mean-square prediction error) of speech sample $s[n]$ in terms of past samples $s[n-k]$ with $k = 1, 2, \ldots, P$. The predicted sample $\hat{s}[n]$ is given by:

$$\hat{s}[n] = \sum_{k=1}^{P} a_k s[n-k] \tag{3.3}$$

with P the required number of past sample of $s[n]$. The prediction error can be formulated as:

$$e[n] = s[n] - \hat{s}[n] = s[n] - \sum_{k=1}^{P} a_k s[n-k] \tag{3.4}$$

To find the predictor coefficients several methods exist, such as the *Covariance* method and the *Autocorrelation* method. In both methods the key to finding the predictor coefficients involves solving large matrix equations.

3.2.2 Mel-Frequency Cepstral Analysis

In contrast to linear predictive coding, Mel-frequency cepstral analysis is a *perceptually motivated* representation. Perceptually motivated representations include some aspect of the human auditory system in their design. In the case of Mel-frequency cepstral analysis, a nonlinear scale, referred to as the Mel-scale, is used that mimics the acoustic range of the human hearing. The Mel-scale can be approximated by:

$$\text{Mel}(f) = 2595 \log_{10}(1 + \frac{f}{700}) \tag{3.5}$$

The process of obtaining feature vectors based on the Mel-frequency is illustrated in figure 3.3. First, the signal is transformed to the spectral domain

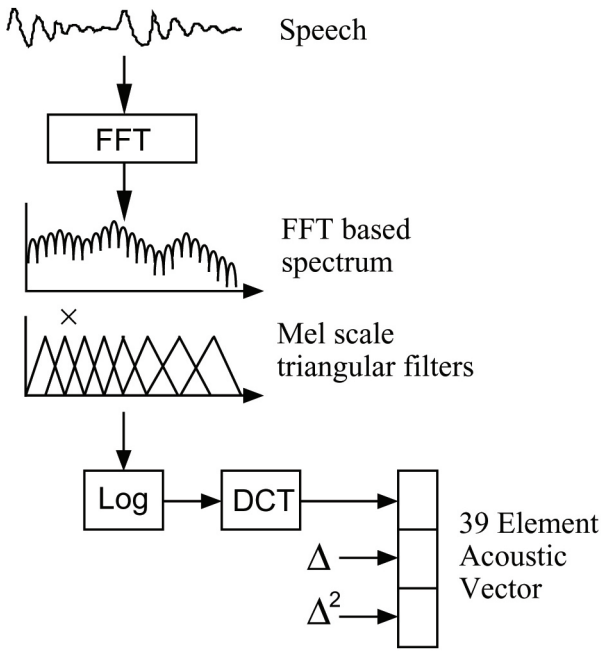

Figure 3.3 Mel-frequency cepstral coefficients.

by a Fourier transform. The obtained spectrum of the speech signal is then smoothed by integrating the spectral coefficients with triangular frequency bins arranged on the non-linear Mel-scale. Next, a log compression is applied to the filter bank output, in order to make the statistics of the estimated speech power spectrum approximately Gaussian. In the final processing stage, a discrete cosine transform (DCT) is applied. It is common for feature vectors derived from Mel-frequency cepstral analysis to contain first-order and second-order differential coefficients besides the static coefficients. Sometimes a measure of the signal energy is included. A typical system usings feature vectors based on *Mel-frequency cepstral coefficients* (MFCCs) can have the following configuration:

- 13th-order MFCC $\mathbf{c}_k$
- 13th-order 1st-order delta MFCC computed from $\triangle\mathbf{c}_k = \mathbf{c}_{k+2} - \mathbf{c}_{k-2}$
- 13th 2nd-order delta MFCC computed from $\triangle\triangle\mathbf{c}_k = \triangle\mathbf{c}_{k+1} - \triangle\mathbf{c}_{k-1}$

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{c}_k \\ \triangle\mathbf{c}_k \\ \triangle\triangle\mathbf{c}_k \end{pmatrix} \tag{3.6}$$

3.3 Hidden Markov Models

In this section the *hidden Markov model* (HMM) will be introduced. The HMM is a powerful statistical method of characterizing data samples of a discrete time-series. Data samples can be continuously or discretely distributed and can

be either scalars or vectors. The HMM has become the most popular method for modeling human speech and is used successfully in automatic speech recognition, speech synthesis, statistical language modeling and other related areas. As an introduction to hidden Markov models, the *Markov chain* will be described first.

3.3.1 The Markov Chain

A Markov chain is essentially a method of modeling a class of random processes, incorporating a limited amount of memory. Let $\mathbf{X} = X_1, X_2, \ldots, X_n$ be a sequence of random variables from a finite discreet alphabet $O = o_1, o_2, \ldots, o_M$. Based on Bayes' rule:

$$P(X_1, X_2, \ldots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i | X_1^{i-1}) \tag{3.7}$$

with $X_1^{i-1} = X_1, X_2, \ldots, X_{i-1}$. The random variables X are said to form a first-order Markov chain provided that

$$P(X_i | X_1^{i-1}) = P(X_i | X_{i-1}) \tag{3.8}$$

Equation 3.7 then becomes

$$P(X_1, X_2, \ldots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i | X_{i-1}) \tag{3.9}$$

Equation 3.8 is referred to as the *Markov assumption*. The Markov assumption states that the probability of a random variable at a given time depends only on its probability at the preceding time. This assumption allows dynamic data sequences to be modeled using very little memory.

If X_i is associated with a state, the Markov chain can be represented by a finite state machine with transitions between states s and s' specified by the probability function

$$P(s|s') = P(X_i = s | X_{i-1} = s') \tag{3.10}$$

With this representation, the Markov assumption is translated into the following: the probability that the Markov chain will be in a particular state at a particular time depends only on the state of the Markov chain at the previous time.

Consider a Markov chain with N states labeled $1, \ldots, N$, with the state at time t denoted by s_t. The parameters of a Markov chain can be described as follows:

$$a_{ij} = P(s_t = j | s_{t-1} = i) \qquad 1 \leq i, j \leq N \tag{3.11}$$

$$\pi_i = P(s_1 = i) \qquad 1 \leq i \leq N \tag{3.12}$$

with a_{ij} the transition probability from state i to state j and π_i the initial probability that the Markov chain will start in state i. The transition and initial probability are bound by the following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad 1 \leq i \leq N \tag{3.13}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{3.14}$$

The Markov chain as described, is also referred to as the *observable* Markov model, as the output of the process is the set of states at each time instance t, where each state corresponds to an observable event X_i. There is a one-to-one correspondence between the observable event sequence $\mathbf{X}$ and the Markov chain states sequence $\mathbf{S} = s_1, s_2, \ldots, s_n$

Figure 3.4 illustrates a simple three-state Markov chain. In the example the three states represent the performance of the stock market in comparison to the previous day. The output symbols are given by $O = \{up, down, unchanged\}$.

state 1 – *up*

state 2 – *down*

state 3 – *unchanged*

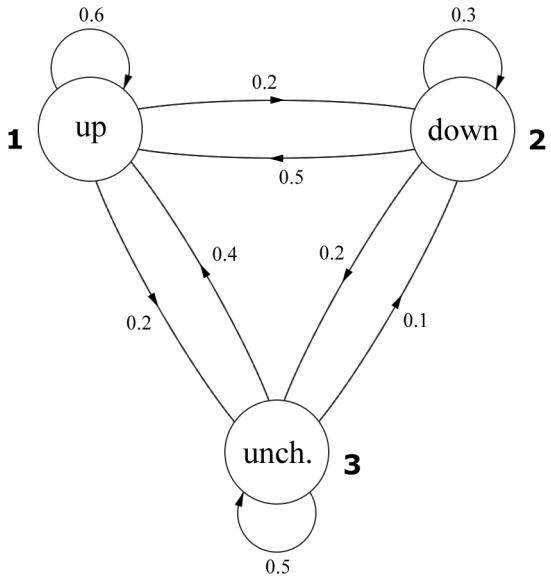

Figure 3.4 A Markov chain example.

The parameters for the stock market example include a state-transition probability matrix

$$A = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.5 \end{bmatrix} \tag{3.15}$$

and an initial probability matrix

$$\pi = (\pi_i)^t = \begin{pmatrix} 0.5 \\ 0.2 \\ 0.3 \end{pmatrix} \tag{3.16}$$

The probability of, for example, five consecutive *up* days, can be found by

$$\begin{aligned} P(5 \text{ consecutive } up \text{ days}) &= P(1,1,1,1,1) = \pi_1 a_{11} a_{11} a_{11} a_{11} \\ &= 0.5 \times (0.6)^4 = 0.0648 \end{aligned} \tag{3.17}$$

3.3.2 Hidden Markov Models

The *hidden Markov model* is an extension of the Markov chain. Instead of each state corresponding to a deterministically observable event, however, the hidden Markov model features a non-deterministic process that generates output observation symbols in any given state. The observation becomes a probabilistic function of the state. In this way the hidden Markov model can be regarded as a double-embedded stochastic process with an underlying stochastic process (the state sequence) that is not directly observable.

A hidden Markov model is essentially a Markov chain where the output observation is a random variable X generated according to an output probabilistic function associated with each state. Figure 3.5 illustrates the stock market example from the previous subsection. Each state can generate all output observations: *up*, *down*, *unchanged*, according to its output probability density function. There is no longer a one-to-one mapping between the observation sequence and the state sequence. For a given observation sequence, the state sequence is not directly observable, hence the naming of *hidden Markov models*.

Formally, a hidden Markov model is defined by:

- $O = \{o_1, o_2, \ldots, o_M\}$—An output observation alphabet. The observation symbols correspond to the physical output of the system being modeled. In the example, the output observation alphabet is the collection of three categories $O = \{up, down, unchanged\}$.

- $\Omega = \{1, 2, \ldots, N\}$—A set of states representing the state space. Here s_t is denoted as the state at time t.

- $\mathbf{A} = \{a_{ij}\}$—A transition probability matrix, where a_{ij} is the probability of taking a transition from state i to state j.

$$a_{ij} = P(s_t = j | s_{t-1} = i) \qquad 1 \leq i, j \leq N \tag{3.18}$$

- $\mathbf{B} = \{b_i(k)\}$—An output probability matrix, with $b_i(k)$ the probability of emitting symbol o_k when state i is entered. Let $\mathbf{X} = X_1, X_2, \ldots, X_t, \ldots$ be the observed output of the HMM. The state sequence $S = s_1, s_2, \ldots, s_t, \ldots$ is not observed and $b_i(k)$ can be rewritten as follows:

$$b_i(k) = P(X_t = o_k | s_t = i) \qquad 1 \leq i \leq N \tag{3.19}$$

- $\pi = \{\pi_i\}$—An initial state distribution with

$$\pi_i = P(s_0 = i) \qquad 1 \leq i \leq N \tag{3.20}$$

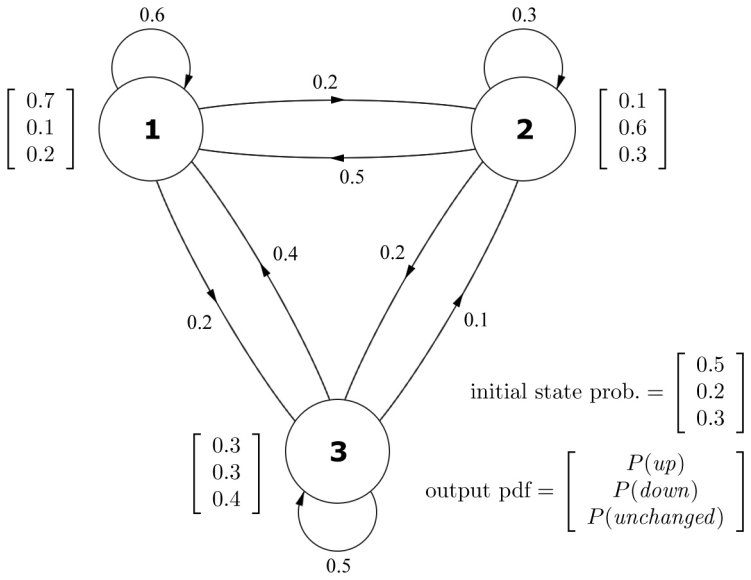

Figure 3.5 A hidden Markov model example.

The following constraints must be satisfied:

$$a_{ij} \geq 0, \qquad b_i(k) \geq 0, \qquad \pi_i \geq 0 \qquad \forall i, j, k \tag{3.21}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{3.22}$$

$$\sum_{k=1}^{M} b_i(k) = 1 \tag{3.23}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{3.24}$$

A complete specification of an HMM thus includes two constant-size parameters N and M, representing the total number of states and the size of observation alphabets, the observation alphabet O and three probability matrices: $\mathbf{A}$, $\mathbf{B}$ and π. The complete HMM can be denoted by

$$\mathbf{\Phi} = (\mathbf{A}, \mathbf{B}, \pi) \tag{3.25}$$

The model described above is a first-order hidden Markov model and is governed by two assumptions. The first is the *Markov assumption* as described for the Markov chain

$$P(s_t|s_1^{t-1}) = P(s_t|s_{t-1}) \tag{3.26}$$

with s_1^{t-1} state sequence $s_1, s_2, \ldots, s_{t-1}$. The second is the *output independence assumption*

$$P(X_t|X_1^{t-1}, s_1^t) = P(X_t|s_t) \tag{3.27}$$

with X_1^{t-1} the output sequence $X_1, X_2, \ldots, X_{t-1}$. The output-independence assumption states that the probability that a particular symbol is emitted at time t depends only on the state s_t and is conditionally independent of past observations.

Given the definition of an HMM above, there are three basic problems that need to be addressed.

1. **The Evaluation Problem** Given a model $\mathbf{\Phi}$ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the probability $P(\mathbf{X}|\mathbf{\Phi})$, i.e. the probability that the model generates the observation?

2. **The Decoding Problem** Given a model $\mathbf{\Phi}$ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the most likely state sequence $\mathbf{S} = (s_0, s_1, s_2, \ldots, s_T)$ that produces the observation?

3. **The Learning Problem** Given a model $\mathbf{\Phi}$ and a sequence of observations, how can the model parameter $\hat{\mathbf{\Phi}}$ be adjusted to maximize the joint probability $\prod_X P(\mathbf{X}|\mathbf{\Phi})$?

To use HMMs for pattern recognition, the *evaluation* problem needs to be solved, which will provide a method to determine how well a given HMM matches a given observation sequence. The likelihood $P(\mathbf{X}|\mathbf{\Phi})$ can be used to calculate the *a posteriori* probability $P(\mathbf{\Phi}|\mathbf{X})$, and the HMM with the highest probability can be determined as the pattern for the best observation sequence. Solving the *decoding* problem will make it possible to find the best matching state sequence given an observation sequence (i.e. the hidden state sequence). This is essential to automatic speech recognition. If the *learning* problem can be solved, model parameter $\mathbf{\Phi}$ can be automatically estimated from training data. The next three subsections will focus in depth on the algorithms involved in solving these three problems.

3.3.3 Evaluating HMMs

The most direct method of calculating the probability $P(\mathbf{X}|\mathbf{\Phi})$ of the observation sequence $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, given the HMM $\mathbf{\Phi}$ is to sum up the probabilities of all possible state sequences:

$$P(\mathbf{X}|\mathbf{\Phi}) = \sum_{\text{all } \mathbf{S}} P(\mathbf{S}|\mathbf{\Phi}) P(\mathbf{X}|\mathbf{S}, \mathbf{\Phi}) \tag{3.28}$$

Basically all possible state sequences $\mathbf{S}$ of length T that generate observation sequence $\mathbf{X}$ are enumerated, after which the probabilities are summed. For a particular state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$, the state-sequence probability in equation 3.28 can be rewritten by applying the Markov assumption:

$$\begin{aligned} P(\mathbf{S}|\mathbf{\Phi}) &= P(s_1|\mathbf{\Phi}) \prod_{t=2}^{T} P(s_t|s_{t-1}, \mathbf{\Phi}) \\ &= \pi_{s_1} a_{s_1 s_2} \ldots a_{s_{T-1} s_T} = a_{s_0 s_1} a_{s_1 s_2} \ldots a_{s_{T-1} s_t} \end{aligned} \tag{3.29}$$

with $a_{s_0 s_1}$ denoting π_{s_1} for simplicity. For the same state sequence S, the joint probability in equation 3.28 can be rewritten by applying the output-

independence assumption:

$$\begin{aligned} P(\mathbf{X}|\mathbf{S},\mathbf{\Phi}) &= P(X_1^T|s_1^T,\mathbf{\Phi}) = \prod_{t=1}^{T} P(X_t|s_t,\mathbf{\Phi}) \\ &= b_{s_1}(X_1)b_{s_2}(X_2)\ldots b_{s_T}(X_T) \end{aligned} \tag{3.30}$$

By substituting equations 3.29 and 3.30 into equation 3.28, equation 3.28 can be rewritten as:

$$\begin{aligned} P(\mathbf{X}|\mathbf{\Phi}) &= \sum_{\text{all } \mathbf{S}} P(\mathbf{S}|\mathbf{\Phi})P(\mathbf{X}|\mathbf{S},\mathbf{\Phi}) \\ &= \sum_{\text{all } \mathbf{S}} a_{s_0 s_1} b_{s_1}(X_1) a_{s_1 s_2} b_{s_2}(X_2)\ldots a_{s_{T-1}s_T} b_{s_T}(X_T) \end{aligned} \tag{3.31}$$

Direct evaluation of equation 3.31 is computationally infeasible, as it requires the enumeration of $O(N^T)$ possible state sequences. However, a simple and efficient algorithm to evaluate equation 3.31 exists. This algorithm is referred to as the *Forward Algorithm* and is described in table 3.1. The basic idea is to store intermediate results and use them for subsequent state-sequence calculations. Let $\alpha_t(t)$ be the probability that the HMM is in state i at time t, having generated partial observation $X_1^t = X_1, X_2, \ldots, X_t$.

$$\alpha_t(i) = P(X_1^t, s_t = i|\mathbf{\Phi}) \tag{3.32}$$

The calculation of $\alpha_t(i)$ can be illustrated in a *trellis*, which is depicted in figure 3.6. This figure illustrates the computation of forward probabilities α in a trellis framework for the stock market example, introduced previously in this section. Inside each node is the forward probability corresponding to each state at time t. Given two consecutive *up* days, the initial forward probabilities α at time $t = 1$ are calculated as follows:

$$\begin{array}{lcllcl} \alpha_1(i) &=& \pi_i b_i(X_1) & & & \\ \alpha_1(1) &=& \pi_1 b_1(X_1) &=& (0.5)\cdot(0.7) &=& 0.35 \\ \alpha_1(2) &=& \pi_2 b_2(X_1) &=& (0.2)\cdot(0.1) &=& 0.02 \\ \alpha_1(3) &=& \pi_3 b_3(X_1) &=& (0.3)\cdot(0.3) &=& 0.09 \end{array} \tag{3.33}$$

Table 3.1 The Forward Algorithm

Algorithm 3.1: The Forward Algorithm

Step 1: Initialization

$\alpha_1(i) = \pi_i b_i(X_1) \qquad 1 \le i \le N$

Step 2: Induction

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}\right] b_j(X_t) \quad 2 \le t \le T; \quad 1 \le j \le N$$

Step 3: Termination

$$P(\mathbf{X}|\mathbf{\Phi}) = \sum_{i=1}^{N} \alpha_T(i)$$

If it is required to end in the final state $P(\mathbf{X}|\mathbf{\Phi}) = \alpha_T(s_F)$

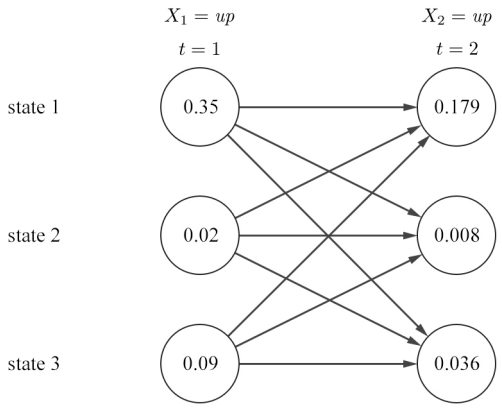

Figure 3.6 Forward trellis computation for the stock market example.

The forward probability at time $t = 2$ for state $j = 1$ is calculated as follows:

$$\begin{aligned}
\alpha_t(j) &= \left[\sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}\right] b_j(X_t) \\
\alpha_2(1) &= \left[\sum_{i=1}^{3} \alpha_1(i)a_{i1}\right] b_1(X_2) \\
&= \Big(\alpha_1(1)a_{11} + \alpha_1(2)a_{21} + \alpha_1(3)a_{31}\Big) b_1(X_2) \\
&= \Big((0.35)\cdot(0.6) + (0.02)\cdot(0.5) + (0.09)\cdot(0.4)\Big)\cdot(0.7) \\
&= (0.256)\cdot(0.7) = 0.1792
\end{aligned} \tag{3.34}$$

The forward probabilities for states $j = 2, 3$ at time $t = 2$ are found similarly:

$$\begin{aligned}
\alpha_2(2) &= \left[\sum_{i=1}^{3} \alpha_1(i)a_{i2}\right] b_2(X_2) \\
&= \Big(\alpha_1(1)a_{12} + \alpha_1(2)a_{22} + \alpha_1(3)a_{32}\Big) b_2(X_2) = 0.0085 \\
\alpha_2(3) &= \left[\sum_{i=1}^{3} \alpha_1(i)a_{i3}\right] b_3(X_2) \\
&= \Big(\alpha_1(1)a_{13} + \alpha_1(2)a_{23} + \alpha_1(3)a_{33}\Big) b_3(X_2) = 0.0357
\end{aligned} \tag{3.35}$$

When the states in the last column have been computed, the sum of all probabilities in the final column is the probability of generating the observation sequence. The complexity of the forward algorithm is $O(N^2T)$ rather than exponential.

3.3.4 Decoding HMMs

The forward algorithm, as discussed in the previous subsection, computes the probability of an HMM generating an observation sequence by summing up

the probabilities of all possible paths. It does not, however, provide the best path (i.e. state sequence). For many applications of HMMs, such as automatic speech recognition, finding the best path is essential. The best path is the state sequence that has the highest probability of being taken while generating the observation sequence. Formally, this is the state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$ that maximizes $P(\mathbf{S}, \mathbf{X}|\mathbf{\Phi})$. An efficient algorithm to find the best state sequence for an HMM exists and is known as the *Viterbi* algorithm. The Viterbi algorithm is described in table 3.2. In practice, this algorithm can also be used to evaluate HMMs, as it offers an approximate solution close to the solution found using the Forward algorithm. The Viterbi algorithm can be regarded as a modified Forward algorithm. Instead of summing up probabilities from different paths coming to the same destination state, the Viterbi algorithm picks and remembers the best path. Let $V_t(i)$ be defined as the probability of the most likely state sequence at time t, which has generated the observation X_1^t and ends in state i.

$$V_t(i) = P(X_1^t, S_1^{t-1}, s_t = i|\mathbf{\Phi}) \tag{3.36}$$

The Viterbi algorithm can be illustrated in a trellis framework similar to the one for the Forward algorithm. Figure 3.7 illustrates the computation of V_t for the stock market example as introduced previously. The number in each cell indicates the best score V_t. The dark lines indicate the best path leading to each cell. Initial values for $V_t(i)$ are calculated as follows:

$$\begin{array}{lcllcll} V_1(i) & = & \pi_i b_i(X_1) & & & & \\ V_1(1) & = & \pi_1 b_1(X_1) & = & (0.5)\cdot(0.7) & = & 0.35 \\ V_1(2) & = & \pi_2 b_2(X_1) & = & (0.2)\cdot(0.1) & = & 0.02 \\ V_1(3) & = & \pi_3 b_3(X_1) & = & (0.3)\cdot(0.3) & = & 0.09 \end{array} \tag{3.37}$$

Table 3.2 The Viterbi Algorithm

Algorithm 3.2: The Viterbi Algorithm

Step 1: Initialization

$V_i(i) = \pi_i b_i(X_i)$ $\quad 1 \le i \le N$

$B_i(i) = 0$

Step 2: Induction

$V_t(j) = \max\limits_{1\le i\le N}\left[V_{t-1}(i)a_{ij}\right] b_j(X_t)$ $\quad 2 \le t \le T; \quad 1 \le j \le N$

$B_t(j) = \arg\max\limits_{1\le i\le N}\left[V_{t-1}(i)a_{ij}\right]$ $\quad 2 \le t \le T; \quad 1 \le j \le N$

Step 3: Termination

The best score $= \max\limits_{1\le i\le N}\left[V_t(i)\right]$

$s_T^* = \arg\max\limits_{1\le i\le N}\left[B_T(i)\right]$

Step 4: Backtracking

$s_t^* = B_{t+1}(s_{t+1}^*)$ $\quad t = T-1, T-2, \ldots, 1$

$\mathbf{S}^* = (s_1^*, s_2^*, \ldots, s_T^*)$ is the best sequence

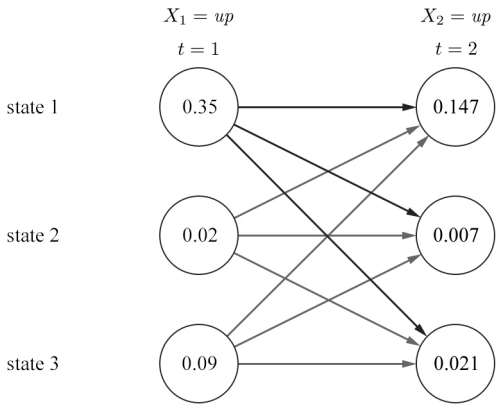

Figure 3.7 Viterbi trellis computation for the stock market example.

Subsequent values for $V_t(i)$ are found as follows:

$$\begin{aligned} V_t(j) &= \max_{1\le i\le N}\Big[V_{t-1}(i)a_{ij}\Big]b_j(X_t) \\ V_2(1) &= \max_{1\le i\le 3}\Big[V_1(i)a_{i1}\Big]b_1(X_2) \\ &= \max_{1\le i\le 3}\Big[V_1(1)a_{11}, V_1(2)a_{21}, V_1(3)a_{31}\Big]b_1(X_2) \\ &= \max_{1\le i\le 3}\Big((0.35)\cdot(0.6), (0.02)\cdot(0.5), (0.09)\cdot(0.4)\Big)\cdot(0.7) = 0.147 \end{aligned} \tag{3.38}$$

$V_2(2)$ and $V_2(3)$ can be found similarly:

$$\begin{aligned} V_2(2) &= \max_{1\le i\le 3}\Big[V_1(i)a_{i2}\Big]b_2(X_2) \\ &= \max_{1\le i\le 3}\Big[V_1(1)a_{12}, V_1(2)a_{22}, V_1(3)a_{32}\Big]b_2(X_2) = 0.007 \\ V_2(3) &= \max_{1\le i\le 3}\Big[V_1(i)a_{i3}\Big]b_3(X_2) \\ &= \max_{1\le i\le 3}\Big[V_1(1)a_{13}, V_1(2)a_{23}, V_1(3)a_{33}\Big]b_3(X_2) = 0.021 \end{aligned} \tag{3.39}$$

Calculation stops at time $t = T$. At this point the best state sequence $\mathbf{S}^*$ can be found using the backtracking pointer $B_t(i)$, which holds the index of the state with the best score $V_t(i)$ in each column of the trellis.

3.3.5 Estimating HMM Parameters

Accurate estimation of model parameters $\mathbf{\Phi} = (\mathbf{A}, \mathbf{B}, \pi)$ is the most difficult of the three problems. The problem can be solved using an iterative algorithm known as the *Baum-Welch* algorithm, sometimes also referred to as the

forward-backward algorithm. The estimation of HMM parameters is a case of *unsupervised learning*, for there is incompleteness of data caused by the hidden state sequence of the HMM. An iterative algorithm for unsupervised learning exists, known as the *expectation maximization* (EM) algorithm, on which the Baum-Welch algorithm is based. It finds model parameter estimates by maximizing the log-likelihood of incomplete data and by iteratively maximizing the expectation of log-likelihood from complete data.

Before the Baum-Welch algorithm can be described, it is necessary to define $\beta_t(i)$, the backward probability as:

$$\beta_t(i) = P(X_{t+1}^T | s_t = i, \mathbf{\Phi}) \tag{3.40}$$

This is the probability of generating partial observation X_{t+1}^T given that the HMM is in state i at time t. The calculation of $\beta_t(i)$ can be performed inductively, as shown in table 3.3. Given $\alpha_t(i)$ and $\beta_t(i)$, it is now possible to define $\gamma_t(i,j)$, the probability of taking the transition from state i to state j at time t, given the model and observation sequence.

$$\begin{aligned} \gamma_t(i,j) &= P(s_{t-1} = i, s_t = j | X_1^T, \mathbf{\Phi}) \\ &= \frac{P(s_{s-1} = i, s_t = j, X_1^T | \mathbf{\Phi})}{P(X_1^T | \mathbf{\Phi})} \\ &= \frac{\alpha_{t-1}(i) a_{ij} b_j(X_t) \beta_t(j)}{\sum_{k=1}^{N} \alpha_T(k)} \end{aligned} \tag{3.41}$$

The calculation of equation 3.41 is illustrated in figure 3.8.

The HMM parameter vector $\mathbf{\Phi}$ can be refined iteratively, by maximizing the likelihood $P(\mathbf{X}|\mathbf{\Phi})$ for each iteration. The new parameter vector derived from $\mathbf{\Phi}$ in the previous iteration, is denoted by $\hat{\mathbf{\Phi}}$. In accordance with the EM algorithm, the following function needs to be maximized:

$$\mathcal{Q}(\mathbf{\Phi}, \hat{\mathbf{\Phi}}) = \sum_{\text{all } \mathbf{S}} \frac{P(\mathbf{X}, \mathbf{S}|\mathbf{\Phi})}{P(\mathbf{X}|\mathbf{\Phi})} \log P(\mathbf{X}, \mathbf{S}|\hat{\mathbf{\Phi}}) \tag{3.42}$$

Equation 3.42 can be rewritten as:

$$\mathcal{Q}(\mathbf{\Phi}, \hat{\mathbf{\Phi}}) = \mathcal{Q}_{\mathbf{a}_i}(\mathbf{\Phi}, \hat{\mathbf{a}}_i) + \mathcal{Q}_{\mathbf{b}_j}(\mathbf{\Phi}, \hat{\mathbf{b}}_j) \tag{3.43}$$

Table 3.3 Calculation of the backward probability.

Algorithm 3.3: Calculating The Backward Probability

Step 1: Initialization

$\beta_T(i) = 1/N$ $\qquad 1 \le i \le N$

Step 2: Induction

$\beta_t(i) = \left[\sum_{j=1}^{N} a_{ij} b_j(X_{t+1}) \beta_{t+1}(j) \right]$ $\qquad t = T-1, \ldots, 1; \quad 1 \le i \le N$

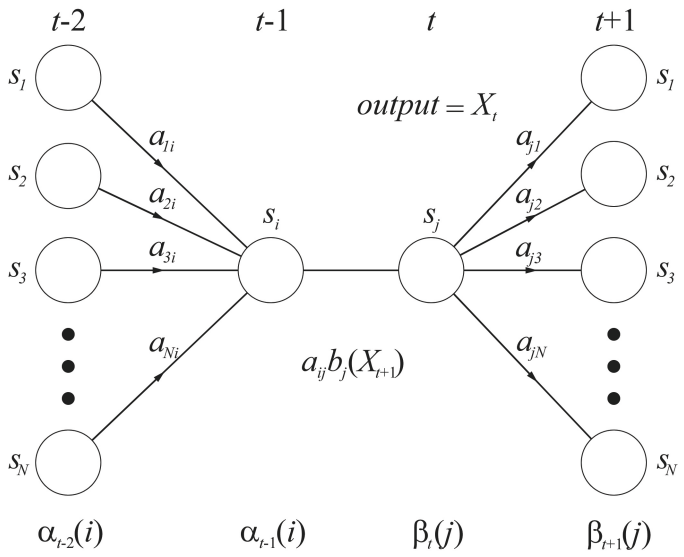

Figure 3.8 Operations required to calculate $\gamma_t(i,j)$.

with,

$$\mathcal{Q}_{\mathbf{a}_i}(\mathbf{\Phi}, \hat{\mathbf{a}}_i) = \sum_i \sum_j \sum_t \frac{P(\mathbf{X}, s_{t-1} = i, s_t = j|\mathbf{\Phi})}{P(\mathbf{X}|\mathbf{\Phi})} \log \hat{a}_{ij} \tag{3.44}$$

$$\mathcal{Q}_{\mathbf{b}_j}(\mathbf{\Phi}, \hat{\mathbf{b}}_j) = \sum_j \sum_k \sum_{t \in X_t = o_k} \frac{P(\mathbf{X}, s_t = j|\mathbf{\Phi})}{P(\mathbf{X}|\mathbf{\Phi})} \log \hat{b}_j(k) \tag{3.45}$$

Equations 3.44 and 3.45 are both of the form:

$$F(x) = \sum_i y_i \log x_i \tag{3.46}$$

which achieves its maximum value at:

$$x_i = \frac{y_i}{\sum_i y_i} \tag{3.47}$$

Thus,

$$\hat{a}_{ij} = \frac{\frac{1}{P(\mathbf{X}|\mathbf{\Phi})} \sum_{t=1}^{T} P(\mathbf{X}, s_{t-1} = i, s_t = j|\mathbf{\Phi})}{\frac{1}{P(\mathbf{X}|\mathbf{\Phi})} \sum_{t=1}^{T} P(\mathbf{X}, s_{t-1} = i|\mathbf{\Phi})} = \frac{\sum_{t=1}^{T} \gamma_t(i,j)}{\sum_{t=1}^{T} \sum_{k=1}^{N} \gamma_t(i,k)} \tag{3.48}$$

$$\hat{b}_j(k) = \frac{\frac{1}{P(\mathbf{X}|\mathbf{\Phi})}\sum_{t=1}^{T} P(\mathbf{X}, s_t = j|\mathbf{\Phi})\delta(X_t, o_k)}{\frac{1}{P(\mathbf{X}|\mathbf{\Phi})}\sum_{t=1}^{T} P(\mathbf{X}, s_t = j|\mathbf{\Phi})} = \frac{\sum_{t \in X_t = o_k}\sum_{i}\gamma_t(i,j)}{\sum_{t=1}^{T}\sum_{i}\gamma_t(i,j)} \quad (3.49)$$

Comparable to the EM algorithm, the Baum-Welch algorithm guarantees a monotonic improvement of the likelihood in each iteration. Eventually the likelihood will converge to a local maximum. Table 3.4 lists the Baum-Welch (Forward-Backward) algorithm. The algorithm, as listed, is based on a single observation sequence, although in practice many observation sequences will be used. The algorithm can easily be generalized to take multiple training observation sequences into account.

Table 3.4 The Forward-Backward Algorithm

Algorithm 3.4: The Forward-Backward Algorithm

Step 1: Initialization
Choose an initial estimate $\mathbf{\Phi}$.

Step 2: E-step
Compute auxiliary function $\mathcal{Q}(\mathbf{\Phi}, \hat{\mathbf{\Phi}})$ based on $\mathbf{\Phi}$.

Step 3: M-step
Compute $\hat{\mathbf{\Phi}}$ according to the re-estimation equations 3.48 and 3.49 to maximize the auxiliary $\mathcal{Q}$-function.

Step 4: Iteration
Set $\mathbf{\Phi} = \hat{\mathbf{\Phi}}$, repeat from step 2 until convergence.

3.4 Acoustic Modeling

This section focuses on the application of hidden Markov models to modeling human speech. First, the selection of appropriate modeling units will be described, after which model topology will be discussed.

3.4.1 Selecting Model Units

When considering using hidden Markov models to model human speech, an essential question is what unit of language to use. Several possibilities exist, such as: words, syllables or phonemes. Each of these possibilities has advantages as well as disadvantages. At a high level, the following criteria need to be considered when choosing an appropriate unit:

- The unit should be *accurate* in representing the acoustic realization in different contexts.
- The unit should be *trainable*. Enough training data should exist to properly estimate unit parameters.

- The unit should be *generalizable*, so that any new word can be derived.

A natural choice to consider is using whole-word models, which have the advantage of capturing the coarticulation effects inherent within these words. When properly trained, word models in small-vocabulary recognition systems yield the best recognition results compared to other units. Word models are both *accurate* and *trainable* and there is no need to be *generalizable*. For large-vocabulary continuous speech recognition, however, whole word models are a poor choose. Given a fixed set of words, there is no obvious way to derive new words, making word models not *generalizable*. Each word needs to be trained separately and thus a lot of training data is required to properly train each unit. Only if such training data exists, are word models *trainable* and *accurate*.

An alternative to using whole-word models is the use of phonemes. European language, such as English and Dutch, typically have between 40 and 50 phonemes. Acoustic models based on phonemes can be trained sufficiently with as little as a few hundred sentence, satisfying the *trainability* criterium. Phoneme models are by default *generalizable* as they are the principle units all vocabulary can be constructed with. Accuracy, however, is more of an issue, as the realization of phonemes is strongly affected by its neighboring phonemes, due to coarticulatory effects such as those described in chapter 2.

Phonetic models can be made significantly more accurate by taking context into account, which usually refers to the immediate left and right neighboring phonemes. This leads to *biphone* and *triphone* models. A *triphone* phoneme model takes into consideration both its left and right neighbor phone thus capturing the most important coarticulatory effects. Unfortunately *trainability* becomes an issue when using triphone models, as there can be as many as $50 \times 50 \times 50 = 125,000$ of them.

3.4.2 Model Topology

Speech is a non-stationary signal that evolves over time. Each state of an HMM has the ability to capture some stationary segment in a non-stationary speech signal. A left-to-right topology thus seems the natural choice to model the speech signal. Transition from left-to-right enable a natural progression of the evolving signal and self-transition can be used to model speech features belonging to the same state. Figure 3.9 illustrates a typical 3-state HMM common to many speech recognition systems. The first state, the entry-state, and the final state, the exit-state are so called *null-states*. These states do not have self loops and do not generate observations. Their purpose is merely to concatenate different models.

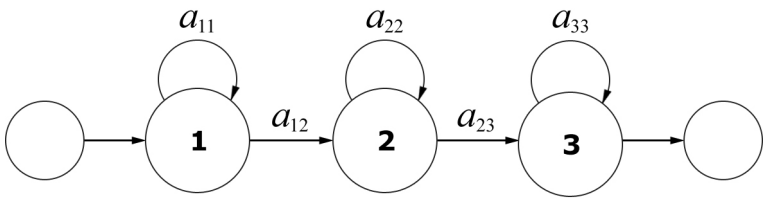

Figure 3.9 Basic structure of a phonetic HMM.

The number of internal states of an HMM can vary depending on the model unit. For HMMs representing a phoneme, three to five states are commonly used. If the HMM represents a word, a significantly larger number of internal states is required. Depending on the pronunciation and duration of the word, this can be 15 to 25 states. More complex transitions between states than the simple topology illustrated in figure 3.9 are also possible. If skipping states is allowed, the model becomes more flexible, but also harder to train properly.

The choice of output probability function $b_j(\mathbf{x})$ is essential to good recognizer design. Early HMM systems used discrete output probability functions in conjunction with vector quantization. Vector quantization is computationally efficient but introduces quantization noise, limiting the precision that can be obtained. Most contemporary systems use parametric continuous density output distributions. Multivariate Gaussian mixture density functions, which can approximate any continuous density function, are popular among contemporary recognition systems. Given M Gaussian mixture density functions:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} c_{jk}\mathcal{N}(\mathbf{x}, \mu_{jk}, \mathbf{\Sigma}_{jk}) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{x}) \tag{3.50}$$

with $\mathcal{N}(\mathbf{x}, \mu_{ij}, \mathbf{\Sigma}_{jk})$, or $b_{jk}(\mathbf{x})$, a single Gaussian density function with mean vector μ_{jk} and covariance matrix $\mathbf{\Sigma}_{jk}$ for state j, M the number of mixture-components and c_{jk} the weight of the k^{th} mixture component, which satisfies:

$$\sum_{k=1}^{M} c_{jk} = 1 \tag{3.51}$$

3.5 Language Modeling

The purpose of the language model is to make an estimation of the probability of a word w_k in an utterance, given the preceding words $W_1^{k-1} = w_1 \dots w_{k-1}$. A popular stochastic language model is the N-gram approach, in which it is assumed that w_k depends only on the preceding $n-1$ words,

$$P(w_k|W_1^{k-1}) = P(w_k|W_{k-n+1}^{k-1}) \tag{3.52}$$

N-grams are very effective in languages where word order is important and strong contextual effects come from near neighbors. N-grams can be computed directly from corpus data, so no formal grammar or linguistic rules are required. The estimation of N-grams can be achieved by a simple frequency count. In the case of *trigrams* ($N = 3$)

$$\hat{P}(w_k|w_{k-1}, w_{k-2}) = \frac{t(w_{k-2}, w_{k-1}, w_k)}{b(w_{k-2}, w_{k-1})} \tag{3.53}$$

where $t(a, b, c)$ is the number of times the trigram *a,b,c* appears in the training data and $b(a, b)$ is the number of times the bigram *a,b* appears.

Unfortunately, when considering V words, there are a total of V^3 potential trigrams, which, for even a modestly sized vocabularies, can easily be a very large number. Many of these will not appear in the training data, or only very few times. The problem is thus one of data sparsity. Several methods have been developed to cope with this problem and will be discussed in detail in the next chapter.

3.6 Decoding

As was described in section 3.1, the aim of the decoder is to find the word sequence $\hat{\mathbf{W}} = w_1, w_2, \ldots, w_m$ with the maximum *a posteriori* probability $P(\mathbf{W}|\mathbf{X})$ for the given acoustic observation $\mathbf{X} = X_1, X_2, \ldots, X_n$. Formulated using Bayes' decision rule:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{X}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{X})} = \arg\max_{\mathbf{W}} P(\mathbf{X}|\mathbf{W})P(\mathbf{W}) \tag{3.54}$$

The unit of acoustic model can be a word model, phoneme model, or some other type. If subwords models are used, they can be concatenated to form word models, according to a pronunciation dictionary or lexicon. The language model $P(\mathbf{W})$, as introduced in the previous section, can be regarded as a network of states, with each state representing a word. If the words are substituted by the correct acoustic models, finding the best word sequence $\hat{\mathbf{W}}$ is equivalent to searching for an optimal path through this network.

In order to make searching for the optimal path efficient, two techniques are commonly employed by search algorithms: *sharing* and *pruning*. Sharing refers to keeping intermediate results, or intermediate paths, avoiding redundant re-computation. Pruning means discarding unpromising paths without wasting time in exploring them further. Search algorithms usually have a *cost* function that needs to be minimized and logarithms are used to avoid extensive multiplication. Equation 3.54 can thus be reformulated as:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \mathcal{C}(\mathbf{W}|\mathbf{X}) = \log\left[\frac{1}{P(\mathbf{X}|\mathbf{W})P(\mathbf{W})}\right] = -\log\left[P(\mathbf{X}|\mathbf{W})P(\mathbf{W})\right] \tag{3.55}$$

As was mentioned in section 3.3, the Viterbi algorithm is used for decoding. Search algorithms based on the Viterbi algorithm have been applied successfully to a wide range of speech recognition tasks. In the next subsection *Time-Synchronous Viterbi beam search* will be discussed.

3.6.1 Time-Synchronous Viterbi Beam Search

When HMMs are used for acoustic models, the acoustic model likelihood can be found using the Forward algorithm, introduced in section 3.3. All possible state sequences must be considered:

$$P(\mathbf{X}|\mathbf{W}) = \sum_{all \text{ possible } s_0^T} P(\mathbf{X}, s_0^T|\mathbf{W}) \tag{3.56}$$

As the goal of decoding is to find the best word sequence, the summation in equation 3.56 can be approximated with a maximization that finds the best state sequence instead of the model likelihood. Equation 3.56 becomes:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{X}|\mathbf{W})P(\mathbf{W}) \cong \arg\max_{\mathbf{W}}\left\{P(\mathbf{W}) \max_{s_0^T} P(\mathbf{X}, s_0^T|\mathbf{W})\right\} \tag{3.57}$$

Equation 3.57 is referred to as the *Viterbi approximation*. The Viterbi search is sub-optimal. In principle the search results using the forward probability can

be different from those using Viterbi, though in practice this is seldom the case. The Viterbi search algorithm completely processes time t before moving on to time $t+1$. At time t, each state is updated by the best score from all states at time $t-1$. This is why the Viterbi search is *time-synchronous*. Furthermore, the Viterbi search algorithm is considered a *breadth-first search* with *dynamic programming*.

As was discussed in section 3.3, the computational complexity of the Viterbi search is $O(N^2T)$, with N the number of HMM states and T the length of the utterance. In order to avoid examining an overwhelmingly large number of possibilities, a heuristic search is required. A heuristic in the Viterbi algorithm is the pruning beam. Instead of retaining all candidates at every time frame, a threshold T is used to keep only a subset of promising candidates. The state at time t with the lowest cost $D_{\min}$ is first identified. Then each state at time t with cost $> D_{\min} + T$ is discarded from further consideration before moving on to time frame $t+1$. The beam search is a simple yet effective method of saving computation with very little loss of accuracy. Combined with time-synchronous Viterbi this leads to a powerful search strategy for large vocabulary speech recognition.

Chapter 4

Key Challenges

Although having grown from a novelty in the research community to a major field of research in many universities and companies alike, many problems related to speech recognition still exist. In a U.S. National Science Foundation funded survey to identify key research challenges related to human language technology [5], the following challenges were identified with regards to automatic speech recognition:

1. Robustness
2. Portability
3. Adaptation
4. Language Modeling
5. Confidence Measure
6. Out-of-Vocabulary Words
7. Spontaneous Speech
8. Prosody
9. Modeling Dynamics

In this chapter an overview will be given of how three of these key research challenges, robustness, adaptation and language modeling, have been addressed in recent years.

4.1 Robustness

Today's most advanced speech recognition systems can achieve very high accuracy rates if trained for a particular speaker, in a particular language and speaking style, in a particular environment and limited to a specific task. It remains a serious challenge however to design a recognition system capable of understanding virtually anyone's speech, in any language, on any topic, in any style in all possible environments. Thus, a speech system trained in a lab with

clean speech may degrade significantly in the real world if the clean speech used does not match the real-world speech.

Variability in the speech signal is a main factor involved in the mismatch between training data and testing, as mentioned in chapter 1. Robustness in speech recognition refers to the need to maintain good recognition in spite of this. Over the years many techniques have been developed in order to obtain robust speech recognition. Figure 4.1 shows a number of these techniques, classified into two levels: the *signal level* and the *model level* [6].

In this section a model of the environment will be presented and some of the techniques involved in making recognition robust will be discussed in detail.

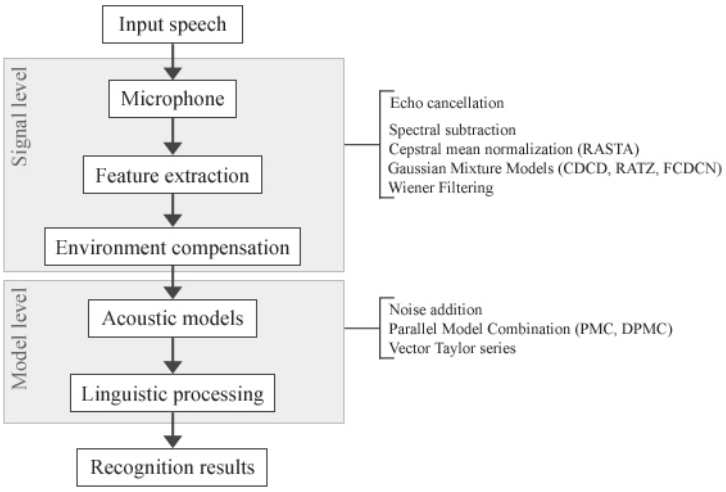

Figure 4.1 Main methods to obtain robust speech recognition.

4.1.1 Variability in The Speech Signal

Variability in the characteristics of the speech signal has three components: *linguistic variability*, *speaker variability* and *channel variability* [16].

Linguistic variability includes the effects of phonetics, phonology, syntax and semantics on the speech signal. When words with different meanings have the same phonetic realization, it becomes impossible for a recognition system to find the correct sequence of words. Consider the example:

> Mr. Wright should write to Mrs. Wright right away about his Ford or four door Honda.

Wright, *write* and *right* are not only phonetically identical, but also semantically relevant. The same is true for *Ford or* and *four door*.

Speaker variability refers to fact that every individual speaker is different and will have a unique speech pattern. This is known as *interspeaker variability*. Speakers can differ in a number of different ways. A person's physical constitution (age, health, size, lung capacity, the size and shape of the vocal tract, etc.) will reflect the characteristics of his speech. Gender also plays an

important role; the pitch of the female voice is in general significantly higher than that of male speakers. Dialect speakers will use different phonemes to pronounce a word than non-dialect speakers or use different allophones of certain phonemes. Further, social environment, education and personal history will all effect the manner in which a person speaks. *Intraspeaker variability* is caused by the fact that even the same person will not be able to exactly reproduce an utterance. This can be caused by a number of reasons such as fatigue and difference in emotional state (irritated, frustrated). When speaking in a noisy environment a person's voice also tends to differ. This is called the *Lombard effect*. In general, in such circumstances vowels will grow longer and much of the voice's energy will shift to higher frequencies.

Channel variability includes the effects of background noise, characteristics of the microphone and channel distortion.

4.1.2 The Acoustic Environment

Three main sources of distortion to speech signals can be distinguished: additive noise, channel distortion and reverberation. Additive noise can be either stationary or nonstationary. In contrast to stationary noise, nonstationary noise has statistical properties that change over time. Examples of stationary noise include fans running in the background, air conditioning, etc. Nonstationary noise includes such things as door slams, radio, TV and other speakers' voices. Channel distortion can be caused by many things: the microphone being used, the presence of electrical filters, properties of speech codecs, local telephone lines, etc. Reverberation is also a main source of distortion. Acoustic waves that reflect off walls and other objects can dramatically alter the signal.

To understand the degradation of the speech signal corrupted by additive noise, channel distortion and reverberation, a model of the environment is presented [12]. This model is illustrated in figure 4.2. The relationship between the clean signal and the corrupted signal in the time domain is given by:

$$y[n] = x[n] \cdot h[n] + v[n] \tag{4.1}$$

with $x[n]$ the clean signal captured at the microphone, $y[n]$ the corrupted signal, $v[n]$ any additive noise present at the microphone and $h[n]$ a linear filter.

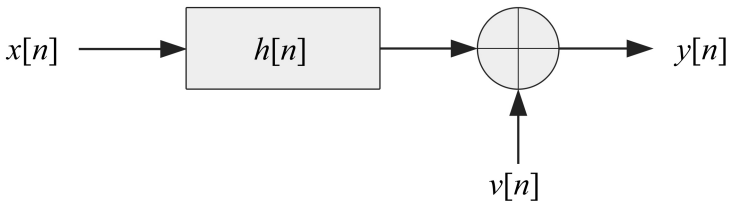

Figure 4.2 A model of the environment.

4.1.3 Adaptive Echo Cancellation

In a standard *full-duplex* speech recognition system, where a microphone is used for a user's voice input, and loudspeakers play back the input signal, it often

occurs that the user's voice is picked up by the microphone as it is output by the loudspeakers and played back again after some delay. This problem can be avoided with a *half-duplex* system that does not listen when a signal is output through the loudspeakers. Systems with *full-duplex* capabilities are desirable, so some form of *echo cancellation* is required. An echo canceling system can be modeled as illustrated in figure 4.3. The return signal $r[n]$ is the sum of speech signal $s[n]$ and the possibly distorted version $d[n]$ of loudspeaker signal $x[n]$.

$$r[n] = d[n] + s[n] \tag{4.2}$$

The purpose of echo cancellation is to remove the echo $d[n]$ from the return signal. This is usually achieved with an adaptive *finite impulse response* (FIR) filter whose coefficients are computed to minimize the energy of the canceled signal $e[n]$. The most common algorithm used to update the FIR coefficients is the *least mean square* (LMS) algorithm, variations of which include the *normalized LMS algorithm*, the *subband LMS algorithm* and the *block LMS* algorithm. The *recursive least squares* algorithm is also common, though computationally more expensive than the LMS algorithm [12].

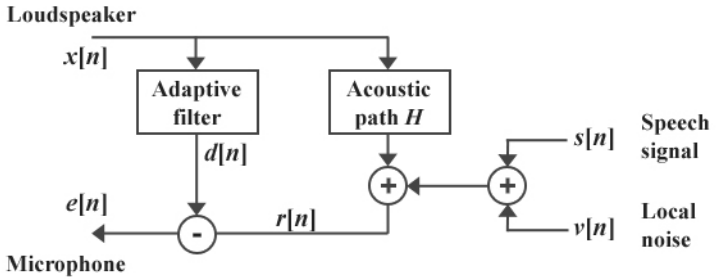

Figure 4.3 Echo canceling application.

4.1.4 Environment Compensation Preprocessing

In order to clean an acoustic signal of additive noise and channel distortion, a number of different techniques can be used. These techniques can also be used to enhance the signal captured from a bad source. As was mentioned in section 3, feature vectors based on Mel-frequency cepstral analysis are common in speech recognition. All techniques presented will be in the context of compensating for the effects of noise on the cepstrum.

A simple and easy to implement method of reducing the effect of additive(uncorrelated) noise in a signal is *spectral subtraction* [12]. The basic idea of *spectral subtraction* is to obtain an estimate of clean speech at the spectral level by subtracting the noise estimation from noisy speech. Consider a clean signal $x[m]$, corrupted by additive noise $n[m]$,

$$y[m] = x[m] + n[m] \tag{4.3}$$

The power spectrum of output $y[m]$ can be approximated by the sum of the power spectra,

$$|Y(f)|^2 \approx |X(f)|^2 + |N(f)|^2 \tag{4.4}$$

It is possible to estimate $|N(f)|^2$ by using the average periodogram over M frames known to be just noise,

$$|\hat{N}(f)|^2 = \frac{1}{M}\sum_{i=0}^{M-1}|Y_i(f)|^2 \tag{4.5}$$

Spectral subtraction supplies an estimate for $|X(f)|$,

$$|\hat{X}(f)|^2 = |Y(f)|^2 - |\hat{N}(f)|^2 = |Y(f)|^2\Big(1 - \frac{1}{\mathrm{SNR}(f)}\Big) \tag{4.6}$$

with,

$$\mathrm{SNR}(f) = \frac{|Y(f)|^2}{|\hat{N}(f)|^2} \tag{4.7}$$

Satisfactory results can be obtained using *spectral subtraction*, though an undesirable effect known as *musical noise* remains. *Musical noise* is noise concentrated in tones of varying and random frequencies. The concept of *spectral subtraction* is constantly being improved upon and many variations exist.

Another powerful and simple technique to increase the robustness of speech recognition is *cepstral mean normalization* (CMN) [12]. The basic idea is to remove special characteristics of the current microphone and room acoustics by subtracting the sample mean of the cepstrum from the input signal. Consider a signal $x[n]$. Its cepstrum can be computed and a set of T cepstral vectors $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T - 1\}$ obtained. The sample mean $\overline{\boldsymbol{x}}$ is given by

$$\overline{\boldsymbol{x}} = \frac{1}{T}\sum_{t=0}^{T-1}\boldsymbol{x}_t \tag{4.8}$$

The normalized cepstrum vector $\hat{\boldsymbol{x}}_t$ can hence be found by

$$\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t - \overline{\boldsymbol{x}} \tag{4.9}$$

Cepstral mean normalization has been found to provide significant robustness when a system is trained on a certain microphone and tested on another. It has also been found that using CMN can reduce error rates for utterances within the same environment as well. This can be explained by the fact that even when using the same microphone, conditions are never exactly the same. Slight differences in room acoustics and differences in the exact distance between mouth and microphone will always exist.

CMN is not suited for real-time systems because a complete utterance is required to compute the cepstral mean. The CMN techniques can be extended by making the cepstral mean $\overline{\boldsymbol{x}}_t$ a function of time [12]. The cepstral mean can be computed by

$$\overline{\boldsymbol{x}}_t = \alpha\boldsymbol{x}_t + (1-\alpha)\overline{\boldsymbol{x}}_{t-1} \tag{4.10}$$

A set of techniques called RASTA also provide a real-time method of cepstral mean normalization.

Gaussian Mixture Models can also be used to make speech signals environmentally robust [12]. The probability distribution of the noisy speech signal $\boldsymbol{y}$ can be modeled as a mixture of K Gaussians by

$$p(\boldsymbol{y}) = \sum_{k=0}^{K-1} p(\boldsymbol{y}|k)P[k] = \sum_{k=0}^{K-1} N(\boldsymbol{y}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)P[k] \tag{4.11}$$

The key is to accurately estimate the parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. Several efficient algorithms for this exist [1] [2]. Other methods of signal preprocessing includes techniques such as Frequency-Domain MMSE from Stereo Data and Wiener Filtering [12].

4.1.5 Environmental Model Adaptation

Robust speech recognition can also be achieved by adapting the HMMs to noisy conditions. Training a large-vocabulary recognition system requires a large amount of training data, which is often not available for specific noise environments. A possible solution is to corrupt the clean speech database with sample sound data taken from a noisy environment and retraining the models. Though it is hard to obtain samples that match the target environment exactly, this adaptation technique nevertheless yields satisfactory results. If the vocabulary is small enough, the retraining can also be done at runtime by keeping the training data in memory.

If the training database is corrupted with noise files of different types and signal-to-noise ratios, a *multistyle training* can be performed. In this case the recognizer will be robust to varying noisy conditions because of the diversity of the training data.

Besides retraining the HMMs by adding environment noise to the speech training database, it is also possible to retrain using feature vectors that have been compensated for distortion effects. The methods described earlier attempt to remove noise and channel effects from the signal during recognition. Given the fact that these techniques are not perfect, it makes sense to consider retraining the HMMs with feature vectors that have been preprocessed.

By far the most popular method of environmental model adaptation is *parallel model combination* (PMC), which is a method to obtain the distribution of noisy speech given the distribution of clean speech and noise as a mixture of Gaussians [12]. Several variants of PMC exist, such as *data-driven parallel model combination*.

Other adaptation methods include using *vector Taylor series*, which attempt to model certain nonlinearities in the speech signal [12].

4.2 Adaptation

In order to make speech recognition systems robust against a continuously changing environment, the use of adaptive techniques is essential. Adaptive techniques are methods to improve the acoustic model accuracy without requiring them to be fully retrained.

Adaptation methods can be either *supervised* or *unsupervised* [28]. In supervised methods the training words or utterances are known to the system in advance, in contrast to unsupervised methods where utterances can be arbitrary. Adaptation methods can be further classified as *on-line* or *off-line*. The on-line methods are used incrementally as the system is in use, working in the

background all the time. Off-line adaptation requires a new speaker to input a certain, fixed amount of training utterances. This process is sometimes referred to as *enrollment* [12], during which a wide range of parameters can be analyzed. Each of these methods may be appropriate for a particular system, however the most useful approach is on-line instantaneous adaptation. This approach is non-intrusive and generally unsupervised; parameters can be modified continuously while the user is speaking.

Two conventional adaptive techniques are *maximum a posteriori probability* (MAP) estimation and *maximum likelihood linear regression* (MLLR) [9]. MAP estimation can effectively deal with data-sparseness problems, as it takes advantage of *a priori* information about existing models. Parameters of pretrained models can be adjusted in such a way that the limited new data will modify the model parameters guided by the a priori knowledge. Formally, an HMM is characterized by a parameter vector Φ. The a priori knowledge about the random vector is characterized by the a priori probability density function $p(\Phi)$. With observation data $\boldsymbol{X}$, the MAP estimate can be expressed as

$$\hat{\Phi} = \arg\max_{\Phi} \left[p(\Phi|\boldsymbol{X})\right] = \arg\max_{\Phi} \left[p(\boldsymbol{X}|\Phi)p(\Phi)\right] \tag{4.12}$$

A limitation of the MAP-based adaptation approach is that a significant amount of new training data is still required, that is, only the model parameters that are actually observed in the adaptation data can be modified.

The most important parameters to adapt, if continuous HMMs are used for acoustic modeling, are the output probability Gaussian density parameters: the mean vector and the covariance matrix. A set of linear regression transformation functions can be used to map the mean and the covariance in order to maximize the likelihood of the adaptation data. The MLLR method is effective for quick adaptation as the transformation parameters can be estimated from a relatively small data set. MLLR is used widely to adapt models to new speakers and new environments. Formally, in the mixture Gaussian density functions, the kth mean vector $\boldsymbol{\mu}_{ik}$ for each state i can be transformed using the equation

$$\tilde{\boldsymbol{\mu}}_{ik} = \boldsymbol{A}_c \boldsymbol{\mu}_{ik} + \boldsymbol{b}_c \tag{4.13}$$

with $\boldsymbol{A}_c$ the regression matrix and $\boldsymbol{b}_c$ an additive bias vector. Besides using MAP and MLLR independently, it is also possible to combine the methods. Satisfactory results can be obtained using this approach.

Another adaptive approach is clustering of similar speakers and environments in the training data [12] and building a set of models for each cluster with minimal mismatch for different conditions. When there is enough training data and enough conditions are represented, significant robustness can be achieved. It is possible to have clusters for different gender, different channels, different speaking styles, etc.

To select the appropriate model in the recognition process, the likelihood of the evaluation speech against all the models can be tested, after which the model with the highest likelihood will be chosen. It is also possible to include the computation of the likelihoods in the recognition system decoder.

Using gender-dependent models can improve the word recognition rate by as much as 10%. More refined clusters can further reduce the error rate. The success of clustering relies on properly anticipating the kind of environment and its characteristics the system will have to deal with.

4.3 Language Modeling

Language modeling in modern speech recognition systems is commonly based on statistics rather than on linguistic theory. Stochastic language modeling (SLM) employs statistical estimation techniques using language training data (i.e. text). The quality of SLM has increased substantially over the past years, as a considerable amount of text of different types has become available online. In this section, two stochastic approaches will be discussed in more detail: *probabilistic context-free grammars* and *N-gram language models*.

4.3.1 Probabilistic Context-Free Grammars

The context-free grammar (CFG), according to Chomsky's formal language theory, is a system of rules to represent an arbitrary sentence as a set of formal symbols. It is defined as

$$G = (V, T, P, S) \tag{4.14}$$

with V and T sets of symbols, P the set of production rules and S the start symbol [12]. The process of mapping a sentence to a set of formal symbols is called parsing. A parser systematically applies the production rules P to a sentence to obtain a parse tree representation of it. The CFG has been around since the 1950s and many parsing algorithms have been developed since. The *bottom-up chart parsing* algorithm is among the state-of-the-art and found in many spoken language understanding systems.

When the CFG is extended to include probabilities for each production rule, a probabilistic CFG (PCFG) is obtained. The use of probabilities allows for a better way to handle ambiguity and becomes increasingly important to correctly applying the production rules when there are many to consider. Formally, the PCFG is concerned with finding the probability of start symbol S generating word sequence $\boldsymbol{W} = w_1, w_2 \dots w_T$, given grammar G

$$P(S \Rightarrow \boldsymbol{W}|G) \tag{4.15}$$

with $\Rightarrow$ symbolizing one or more parsing steps. To determine the probabilities of each rule in G, a training corpus is used. The simplest approach is to count the number of times each rule is used in a corpus containing parsed sentences. The probability of a rule $A \rightarrow \alpha$ occurring is denoted as $P(A \rightarrow \alpha|G)$. If there are m rules for tree node $A : A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots A \rightarrow \alpha_m$, the probability of these rules can be estimated by

$$P(A \rightarrow \alpha_j|G) = C(A \rightarrow \alpha_j) / \sum_{i=1}^{m} C(A \rightarrow \alpha_i) \tag{4.16}$$

with $C(\dots)$ the number of times each rule is used. The key to PCFG is the correct estimation of the production rule probabilities and many sophisticated estimation techniques have been developed.

4.3.2 N-gram Language Models

As mentioned earlier, a language model can be formulated as a probability distribution $P(\boldsymbol{W})$ over word strings $\boldsymbol{W}$ that reflect how frequently $\boldsymbol{W}$ occurs

as a sentence. $P(\boldsymbol{W})$ can be expressed as

$$\begin{aligned} P(\boldsymbol{W}) &= P(w_1, w_2, \ldots, w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\cdots P(w_n|w_1, w_2, \ldots, w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}) \end{aligned} \tag{4.17}$$

with $P(w_i|w_1, w_2, \ldots, w_{i-1})$ the probability that w_i will follow, given word sequence $w_1, w_2, \ldots, w_{i-1}$. In reality, $P(w_i|w_1, w_2, \ldots, w_{i-1})$ is impossible to estimate. For a vocabulary size V, there are a total of V^{i-1} possible histories $w_1, w_2, \ldots, w_{i-1}$, most of which are unique or occur only a few times.

The N-gram language model assumes the probability of the occurrence of a word depends only on the $N-1$ previous words. For instance, in a *trigram* model $P(w_i|w_{i-2}, w_{i-1})$ the word depends only on the previous two words. *Unigram* and *bigram* language models can be expressed similarly.

The estimation of $P(w_i|w_{i-2}, w_{i-1})$ is as straightforward as counting how often the sequence (w_{i-2}, w_{i-1}, w_i) occurs in a given training corpus and normalizing by the number of times sequence (w_{i-2}, w_{i-1}) occurs

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} \tag{4.18}$$

with $C(\ldots)$ the frequency count of pair (w_{i-2}, w_{i-1}) and triplet (w_{i-2}, w_{i-1}, w_i). Consider the example sentence: *John read her a book. I read a different book. John read a book by Mulan.* The symbol <*s*> marks the beginning of a sentence and </*s*> the end of a sentence. $P(John\ read\ a\ book)$ can be found by

$$\begin{aligned} P(John|\ <s>) &= \frac{C(<s>, John)}{C(<s>)} = \frac{2}{3} \\ P(read|John) &= \frac{C(John, read)}{C(John)} = \frac{2}{2} \\ P(a|read) &= \frac{C(read, a)}{C(read)} = \frac{2}{3} \\ P(book|a) &= \frac{C(a, book)}{C(a)} = \frac{1}{2} \\ P(</s>\ |book) &= \frac{C(book, </s>)}{C(book)} = \frac{2}{3} \end{aligned}$$

Thus,

$$\begin{aligned} &P(John\ read\ a\ book) \\ &= P(John|\ <s>)P(read|John)P(a|read)P(book|a)P(</s>\ |book) \\ &\approx 0.148 \end{aligned} \tag{4.19}$$

In this example the training data is extremely limited. Many new sentences will have zero probability even though they are quite reasonable. Also, the N-gram model is essentially blind to grammar. If N is small, grammatically incorrect sentences can still be assigned high probabilities.

Data sparseness is a key problem in N-gram modeling. If the training corpus is too small many possible word successions will not be observed, resulting in

very small probabilities. Consider *Mulan read a book* from the previous example

$$P(read|Mulan) = \frac{C(Mulan, read)}{\sum_{w} C(Mulan, w)} = \frac{0}{1} \tag{4.20}$$

leading to $P(Mulan\ read\ a\ book) = 0$. Most state-of-the-art speech recognition systems use some form of *smoothing* to handle this problem [12] [19]. In essence *smoothing* is a technique to make distributions flatter, that is, adjusting low and zero probabilities upward and high probabilities downward.

A simple smoothing technique is to pretend each bigram occurs once more often than it actually does

$$P(w_i|w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{\sum_{w_i}(1 + C(w_{i-1}, w_i))} = \frac{1 + C(w_{i-1}, w_i)}{V + \sum_{w_i} C(w_{i-1}, w_i)} \tag{4.21}$$

with V the size of the vocabulary. Considering the previous example V is the set of all occurring words, with $V = 11$, including <s> and </s>. The probability of the sentence *John read a book* becomes

$$\begin{aligned} &P(John\ read\ a\ book) \\ &= P(John|\ <s>)P(read|John)P(a|read)P(book|a)P(</s>\ |book) \\ &\approx 0.00035 \end{aligned} \tag{4.22}$$

The probability of sentence *Mulan read a book* becomes

$$\begin{aligned} &P(Mulan\ read\ a\ book) \\ &= P(Mulan|\ <s>)P(read|Mulan)P(a|read)P(book|a)P(</s>\ |book) \\ &\approx 0.000084 \end{aligned} \tag{4.23}$$

Both estimations are much more reasonable than the initial maximum likelihood estimates. In general, most existing smoothing algorithms can be described with the following equation

$$\begin{aligned} &P_{smooth}(w_i|w_{i-n+1} \dots w_{i-1}) \\ &= \begin{cases} \alpha(w_i|w_{i-n+1} \dots w_{i-1}) & \text{if } C(w_{i-n+1} \dots w_i) > 0 \\ \gamma(w_{i-n+1} \dots w_{i-1}) P_{smooth}(w_i|w_{i-n+2} \dots w_{i-1}) & \text{if } C(w_{i-n+1} \dots w_i) = 0 \end{cases} \end{aligned} \tag{4.24}$$

If an N-gram has a nonzero count the distribution $\alpha(w_i|w_{i-n+1} \dots w_{i-1})$ is used. Otherwise, a *backoff* occurs to the lower-order N-gram distribution $P_{smooth}(w_i|w_{i-n+2} \dots w_{i-1})$ with $\gamma(w_{i-n+1} \dots w_{i-1})$ a scaling factor to make the conditional distribution sum to one.

Algorithms in this framework are called *backoff models*. The most popular smoothing technique is *Katz smoothing* [12]. Other smoothing techniques are *Good-Turing estimates* and *Kneser-Ney bigram smoothing* [12].

Many other language modeling methods exist [19] [28]. Adaptive language models include *cache language models*, *topic-adaptive models* and *maximum entropy models*. Models also exist based on decision trees and CART-style algorithms. Other models include *trigger models*, *trellis models*, *history models* and *dependency models*.

Chapter 5

The Hidden Markov Model Toolkit

The *Hidden Markov Model Toolkit* (HTK) is a collection of programming tools for creating and manipulating hidden Markov models (HMMs). The HTK is primarily intended for speech recognition research, though can be used to create HMMs that model any time series.

The HTK was developed at the *Speech Vision and Robotics Group* of the Cambridge University Engineering Department (CUED) to build large vocabulary speech recognition systems. All rights to sell HTK were acquired by Entropic Research Laboratory Inc. in 1993 and full HTK development was transferred to the Entropic Cambridge Research Laboratory Ltd, when it was established in 1995. Microsoft bought Entropic in 1999 and licensed HTK back to CUED in 2000. Microsoft retains the copyright to the HTK code, though it is freely available for research purposes.

This chapter is intended to provide an overview of the HTK. In the following sections the software architecture of the HTK will be described and an outline will be given of the HTK tools and the way they are used to construct and test HMM-based speech recognizers.

5.1 HTK Software Architecture

Essentially, the HTK consists of a number of tools, which realize much of their functionality through a set of library modules. These modules are common across the tools and ensure that each tool interfaces with the outside world in exactly the same way. They also provide access to commonly used functions. The software architecture of HTK is illustrated in figure 5.1. User I/O and interaction with the operating system is controlled by module *HShell* and memory management is controlled by *HMem*. Math support is provided by *HMath* and signal processing operations are provided by *HSigP*.

Each of the files used by HTK has a dedicated interface module. *HLabel* provides the interface for label files, *HLM* for language model files, *HNet* for networks and lattices, *HDict* for dictionaries, *HVQ* for VQ codebooks and *HModel* for HMM definitions.

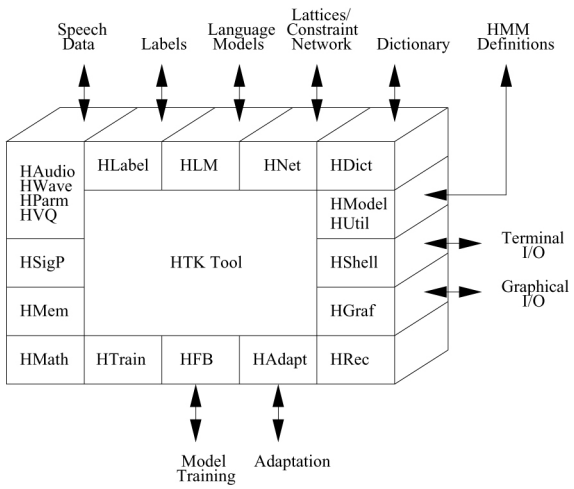

Figure 5.1 HTK software architecture.

Speech I/O is controlled by *HWave* at the waveform level and by *HParm* at the parameter level. These modules support multiple types of audio data. Direct audio input is supported by *HAudio* and simple graphics is provided by *HGraf*. *HUtil* provides functionality for manipulating HMMs while *HTrain* and *HFB* provides support for the HTK training tools. *HAdapt* provides support for the HTK adaptation tools. Finally, *HRec* contains the main recognition processing functions.

Figure 5.2 shows an example of how to run a typical HTK tool. All HTK tools are run from a system command prompt and have optional and required arguments. Optional arguments are prefixed by a minus sign and can have real numbers, integers or string values associated with them. If an option name is a capital letter, it is common across all the HTK tools. In the example the **-T** option indicates the required level of tracing and the **-S** option indicates a script file will be used to supply the tool with the required input files. The HTK text-based command prompt interface has several benefits: it allows shell scripts to control tool execution, which is useful when building large-scale systems that require many files, and it allows details of system development and experiments to be documented easily.

```
HVite -S lpcfiles.lst -i labels.mlf -T 01 -o S -w sabw0001.slf HMMList.txt
```

Figure 5.2 Running an HTK tool.

5.2 The Toolkit

In this section the HTK tools will be described. The tools are divided into four categories that correspond with the three main phases involved in building a

speech recognizer. These are:

1. Data preparation
2. Training
3. Evaluation (Testing & Analysis)

The various HTK tools and the processing phases are illustrated in figure 5.3.

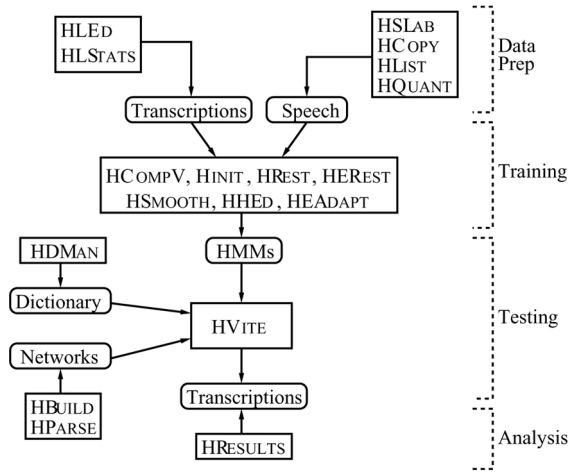

Figure 5.3 HTK processing phases.

5.2.1 Data Preparation Tools

In order to build a speech recognizer a set of speech data files and associated transcriptions are required. A typical database of audio files, referred to as a *corpus*, will contain speech data recorded by many different speakers and is often quite large. Before the corpus can be used to train HMMs it must be converted to an appropriate parametric form and the transcriptions must be converted to the correct format and use the required labels. HTK provides the *HSlab* tool to record audio data and manually annotate it with appropriate transcriptions.

To parameterize audio *HCopy* is used. Essentially, *HCopy* performs a copy operation on an audio file and converts it to the required parametric form while copying. Besides copying the whole file, *HCopy* allows extraction of relevant segments and concatenation of files by specifying appropriate configuration parameters. The tool *HList* can be used to check the contents of speech files and parametric conversion.

Transcriptions usually need some further preparing, as the original source transcriptions will not be exactly as required, for example, because of differences in used phoneme sets. To convert transcriptions to HTK label format, *HLed* can be used. *HLed* is a script-driven label editor and can output transcription files to a single *Master Label File* (MLF).

Other data preparation tools include *HLStats*, which can gather and display statistic on label files and *HQuant*, which can be used to build a VQ codebook for building discrete probability HMM systems.

5.2.2 Training Tools

The next step in building a speech recognizer is to define the topology of each HMM in a prototype definition. The HTK allows HMMs to be built with arbitrary topology. HMM prototype definitions are stored as text files and can be edited with a simple text editor. The prototype definition is intended only to specify the overall characteristics and topology of the HMM, as actual parameters will be computed by the training tools. Sensible values must be chosen for the transition probabilities, but the training process is very insensitive to these. A simple strategy is to give all transition probabilities equal values.

The training of the HMMs takes place in a number of stages, as illustrated in figure 5.4. The first stage is to create an initial set of models. If there is some training data available for which the phone boundaries have been transcribed, then this can be used as *bootstrap data*. In this case, the tools *HInit* and *HRest* provide *isolated word* training using the bootstrap data. The HMMs are generated individually. *HInit* read in all the bootstrap data and *cuts out* examples of the required phone, after which an initial set of parameters values is computed iteratively using a *segmental k-means* procedure. On the first iteration, training data is segmented uniformly, each model state is matched with corresponding data segments and means and variances are estimated. In further iterations uniform segmentation is replaced by Viterbi alignment. After *HInit* has computed the initial parameter values, they are further re-estimated by *HRest*. *HRest* also uses the bootstrap data, but the segmental k-means procedure is replaced by Baum-Welch re-estimation. If there is no bootstrap data available a *flat start*

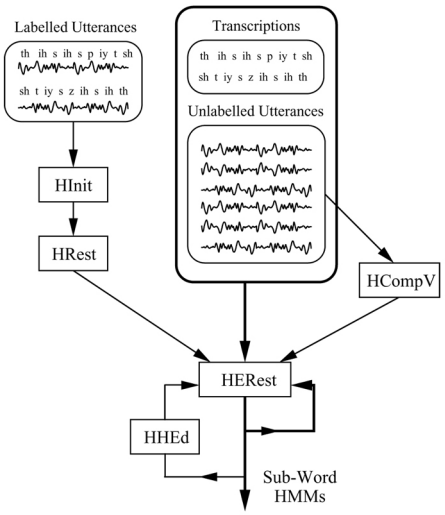

Figure 5.4 Training HMMs.

can be made. In this case HMMs are initialized identically and have state means and variances equal to the global speech mean and variance. The tool *HCompV* can be used for this.

Once an initial set of models has been created, the tool *HERest* is used to perform an *embedded training* using the entire training set. *HERest* performs a single Baum-Welch re-estimation of the whole set of HMM phone models simultaneously. For each training utterance, the corresponding phone models are concatenated and then the forward-backward algorithm is used to gather the relevant statistics of state occupation, means, variance, etc., for each HMM in the sequence. After all training data has been processed, the accumulated statistics are used to compute re-estimates of the HMM parameters. *HERest* is the main HTK tool and many options, such as pruning, can be set.

The HTK allows HMMs to be refined incrementally. Typically, single Gaussian, context-independent models are created first. These can then be iteratively refined by expanding them to include context-dependency (e.g. biphones, triphones) and use multiple mixture component Gaussian distributions. The tool *HHed* can be used to clone models into context-dependent sets and increment the number of mixture components. Using the tools *HEAdapt* and *HVite* performance for specific speakers can be improved by adapting the HMMs using a small amount of adaptation data.

One of the biggest problems in creating context-dependent HMMs is insufficiency of training data. The more complex the model set, the more training data is required, so that a balance must be struck between complexity and available data. This balance can be achieved by tying parameters together, which allows data to be pooled in order to robustly estimate shared parameters. HTK also supports tied mixture and discrete probability systems. The tool *HSmooth* can be used in these cases to address data insufficiency.

5.2.3 Testing Tools

The recognition tool provided by the HTK is called *HVite*. *HVite* uses an algorithm called the *token passing algorithm* to perform Viterbi-based speech recognition. As input, *HVite* requires a network describing the allowable word sequences, a dictionary defining how each word is pronounced and a set of HMMs. *HVite* will convert the word network to a phone network and attach the appropriate HMM definition to each phone instance, after which recognition can be performed on direct audio input or on a list of stored speech files.

The word network required by *HVite* can be a simple word loop or a finite-state task grammar represented by directed graphs. In a simple word loop network any word can follow any other word and bigram probabilities are normally attached to the word transitions. Network files are stored in HTK standard lattice format, which is text-based and can be edited with any text-editor. HTK provides two tools to assist in building the networks: *HBuild* and *HParse*. *HBuild* allows the creation of sub-networks that can be used in higher level networks and can facilitate the generation of word loops. *HParse* is a tool that can convert networks written in a higher level grammar notation to HTK standard lattice format. The higher level grammar notation is based on the Extended Backus Naur Form (EBNF).

To see examples of the possible paths contained within a network, the tool *HSGen* can be used. *HSgen* takes a network as input and will randomly traverse

it and output word strings. These strings can be inspected to confirm the network meets its design specifications.

The construction of large dictionaries can involve merging different sources and performing various transformations on these sources. The tool *HDMan* can assist in this process.

5.2.4 Analysis Tools

Analysing an HMM-based recognizer's performance is usually done by matching a set of transcriptions output by the recognizer with correct reference transcriptions. The tool *HResults* can perform this comparison. It uses dynamic programming to align the two transcriptions and counts substitution, deletion and insertion errors. Optional parameters can be set to ensure that the *HResults* output is compatible with standards set by the U.S. National Institute of Standards and Technology (NIST). *HResults* can also provide speakers-by-speaker breakdowns, confusion matrices and time-aligned transcriptions.

Chapter 6

Developing Acoustic Models

In this chapter the development of acoustic models for the Dutch language is described. Several model sets will be presented: monophone phoneme models, biphone phoneme models, word digit models and word alphabet models.

The target environment of the acoustic models is embedded systems, such as PDAs and car navigation systems. This operating environment imposes several restrictions on the acoustic models. Computational time and memory are limited, thus the model sets need to be small and recognition time must be short. These requirements are reflected in the nature of the acoustic models. Biphones are used instead of triphones, as the number of models will be smaller, and a single Gaussian output probability function is used instead of a more sophisticated distribution.

With regard to the operating environment, robustness of the acoustic models is also essential. A number of techniques are employed to achieve this. First, the models are trained with speech files to which noise has been added. If the training noise accurately matches the noise conditions of the operating environment, model performance in noisy conditions can improve significantly. Also, during acoustic analysis, two environmental compensation techniques are applied to enhance the speech files.

6.1 Overview

The development of acoustic models is performed in three phases: *data preparation*, *training* and *evaluation*. These are illustrated in figure 6.1. In the data preparation phase, all data relevant to model training is selected and preprocessed as required. The core data set is the speech corpus, which contains speech samples recorded by many different speakers. Besides the corpus, there are a number of files and parameters that need to be set up. The following steps constitute the data preparation phase:

1. Selection and categorization of speech data from the Amsterdam corpus.

2. Resampling and noise mixing of speech data.

3. Acoustic analysis of speech data (feature vector extraction).
4. Preparation of pronunciation dictionary.
5. Definition of HTK subwords.
6. Preparation of training network and training label files.
7. Preparation of HMM prototypes.

Data preparation is discussed in section 6.3. The second phase of acoustic model development is the training phase. The training of the acoustic models is discussed in section 6.4 and consists of the following steps:

1. Training of initial models.
2. Ten iterations of Baum-Welch re-estimation.

After model training, the models need to be evaluated and results need to be analyzed. Model evaluation is discussed in section 6.5. The following steps constitute the evaluation phase:

1. Preparation of evaluation network and evaluation labels.
2. Perform recognition on evaluation data.
3. Analysis of results.

After monophone phoneme models have been trained and evaluated, they are expanded into context dependent models. This is discussed in section 6.6. The final section of this chapter is concerned with the development of word digit models and word alphabet models. First, the development environment is described.

Figure 6.1 Three phases of model development.

6.2 Visual Acoustic Model Builder

The *Visual Acoustic Models Builder* (VAMB) is a framework around the HTK designed to facilitate and simplify the development of acoustic models. In essence, VAMB is a set of tools that manipulate training data and model files, combined with a set of scripts that run the HTK commands. The VAMB is illustrated in figure 6.2.

The core of the VAMB is a training configuration file, which contains all the parameters relevant to the training of a particular model set. These parameters include:

- The type of acoustic models to train. Monophone, biphone, phonemic based, word based, etc.
- The categories and location of the training data.
- The training data sample frequency, the type of noise and the signal-to-noise ratio.
- The location of label files, network files and other model configuration files.
- Various other parameters, such as the number of *HERest* iterations and its update flags.

The configuration file is a simple text file which can be edited by hand or using the *MakeModelWizard*, a graphical user interface to the VAMB. A screenshot of the *MakeModelWizard* and a configuration file used to create monophone phoneme models are provided in appendix B.

The model creation process is controlled by 21 Perl scripts that use the parameters specified in the configuration file. The scripts are designed to automate a number of mundane tasks related to the training of acoustic models with HTK, such as the creation of folders, copying files, etc. The scripts also

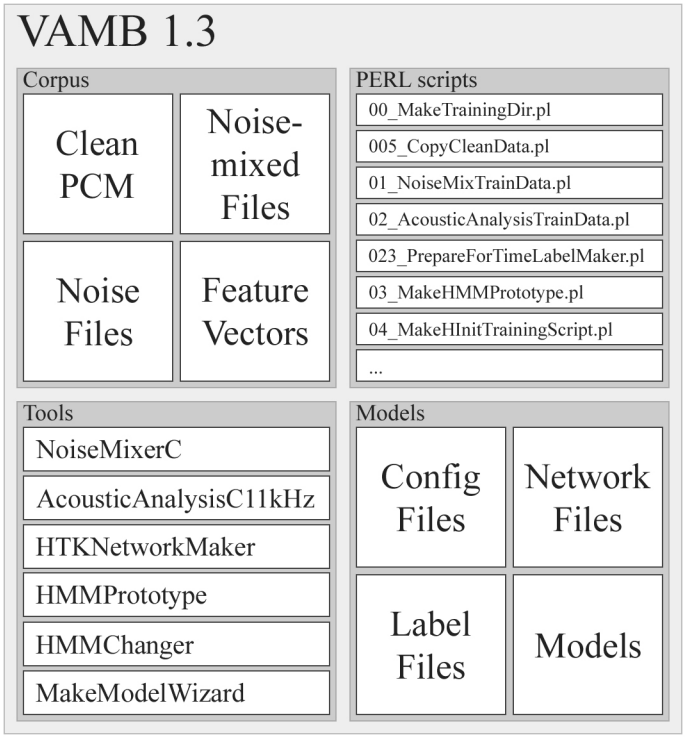

Figure 6.2 VAMB overview.

manage the HTK tools, which usually have long lists of input files and require many parameters to be set up. There are also a number auxiliary scripts within the VAMB framework, designed to perform various other tasks related to the model training process. An overview of the scripts and their purpose can be found in appendix B.

Besides the scripts, VAMB consists of several tools, some of which are shown in figure 6.2. The most important tools will be described in the relevant sections of this chapter.

VAMB was first developed in 2001 to assist in the creation of acoustic models with HTK. Over the years it has been revised, updated and added to.

6.3 Data Preparation

This section describes the steps required to prepare the data that is used to train the Dutch acoustic models.

6.3.1 Corpus Information

The corpus of speech data used to train the Dutch acoustic models consists of studio-quality recordings made in Amsterdam by 150 speakers. The speech audio data is stored as 16-bit, mono, Intel PCM files (waveforms), sampled at 48 kHz.

Recording was done by 75 native Dutch speakers of each gender, originating from different parts of the country, aged between 18 and 55. The speakers are divided into several groups and the utterances are divided into several categories as is illustrated in Table 6.1. Table 6.1 also shows the number of utterances per category and the total number of utterances spoken by each speaker. The corpus contains the following categories:

SABW (Specified Acevet Balanced Word) Mainly single word utterances with a balanced phoneme distribution.

SALW (Specified Allophon Word) Single word utterances containing all phonemes at least once.

SFQW (Specified Frequent Word) A selection of words common in the Dutch language.

EVW (Evaluation Word) A selection of words chosen for system evaluation.

CSD (Common Single Digit) Single digits between 0 and 9, 10 to 14 and tens from 10 to 90.

CCD (Common Connected Digit) Five digits spoken in sequence.

CSA (Common Single Alphabet) The Dutch alphabet.

SCA (Specified Connected Alphabet) Sequences of five letters.

SCD (Specified Connected Digit) Sequences of five numbers in the range 10 to 99.

Categories are either *common* or *specified*. *Common* categories contain utterances spoken by all speakers in all groups, while *specified* categories contain utterances that are unique to a particular speaker group.

The five speaker groups A, B, C, D and E each contain 15 speakers and are distinguished by difference in utterances from the *specified* categories and by

Table 6.1 Dutch language corpus.

ID		SABW	SALW	SFQW	EVW	CSD	CCD	CSA	SCA	SCD
	Total	Specified Word				Common			Specified	
A FH	453	0	0	0	145	25	26	29	100	128
A LH	453	0	0	0	145	25	26	29	100	128
B	575	327	39	30	0	25	26	29	35	64
C	575	327	39	30	0	25	26	29	35	64
D	575	327	39	30	0	25	26	29	35	64
E	574	327	39	30	0	25	26	29	34	64
	Total	1308	156	120	145	100	104	116	139	256
A		SABWA	SALWA	SFQWA	EVW	CSD	CCD	CSA	SCAA	SCDA
B		SABWB	SALWB	SFQWB		CSD	CCD	CSA	SCAB	SCDB
C		SABWC	SALWC	SFQWC		CSD	CCD	CSA	SCAC	SCDC
D		SABWD	SALWD	SFQWD		CSD	CCD	CSA	SCAD	SCDD
E		SABWE	SALWE	SFQWE		CSD	CCD	CSA	SCAE	SCDE

difference in speaking style. The speaking style relates primarily to the speed at which the utterances are spoken. Six styles are distinguished in the corpus, in a range between *normal* and *fast*.

The corpus is divided into training and evaluation speakers. Utterances spoken by speakers from groups B, C, D and E are used in the training of the acoustic models, while utterances spoken by speakers from group A are used for evaluating the models. No utterances from the **SABW**, **SALW** and **SFQW** categories are spoken by speakers in group A. Utterances from the evaluation category **EVW** are spoken instead.

The corpus was designed specifically to train models for speech recognition in car navigation systems and similar command based applications. This is reflected in the nature of the recordings, which are mainly short words of one or two syllables. There are also many recordings of digits.

Associated with the corpus is a *corpus list* file that contains a list of the audio files and the utterances spoken within that file.

6.3.2 Resampling and Noise Mixing

Several operations are performed on the PCM speech data files before they are used to train the acoustic models, as illustrated in figure 6.3. First the data is *resampled*. The original files are sampled at 48 kHz, yet in the target environment audio input is frequently sampled at lower rates. For car navigation systems the audio data is resampled at 11 kHz. For mobile telephone applications a sample rate of 8 kHz is used.

The next step is *noise mixing* of the audio data. A relatively simple method to create robust acoustic models is to train them with audio data that has been mixed with certain kinds of noise, as discussed in section 4.1. The type of noise that will be used depends on the target environment. For car navigation systems, car noise data will be mixed with the speech files. Though the car noise data is recorded in a certain vehicle, with a certain engine type and other specific acoustic characteristics, the addition of this noise nevertheless increases the performance of the acoustic models when applied in other car environments. For other environments, noise data recorded in an exhibition hall is used. This noise data is referred to as *Booth noise*. The speech data and the noise can be mixed at different signal-to-noise ratios, ranging from -5 dB to 20 dB. Both noise mixing and resampling is performed using the tool *NoiseMixerC*.

The final step in the preparation of the speech data files is the acoustic analysis.

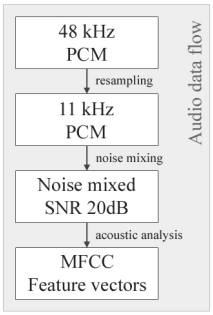

Figure 6.3 Audio data flow.

6.3.3 Acoustic Analysis

The purpose of the acoustic analysis is to transform the PCM speech data into a parameterized form as was discussed in section 3.2. The discrete feature vectors extracted from the waveform data consist of the following components:

- 10 static Mel-frequency cepstrum coefficients (MFCCs)
- 10 first order Mel-frequency regression coefficients (referred to as delta coefficients)
- delta log of the signal energy

In the HTK these components are called *streams*. The process of obtaining the 21-dimensional acoustic feature vectors consists of three main stages, illustrated in figure 6.4. In the first stage two filters are applied to the signal in the time domain. The first, a DC-cut filter removes certain electrical properties from the signal. The second, a pre-emphasis (or high-pass) filter is used to boost the higher frequencies.

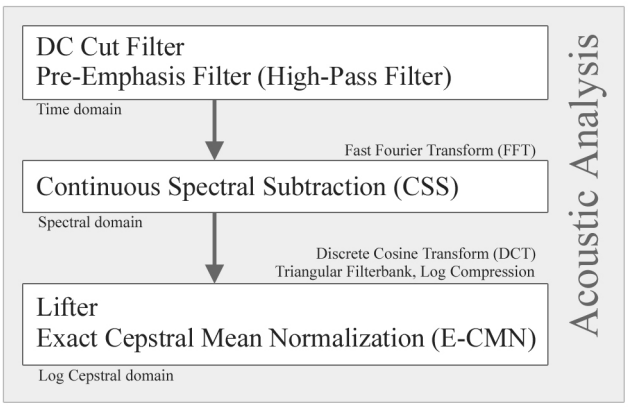

Figure 6.4 Acoustic analysis.

In the second stage a 256 point Fast Fourier Transform is applied to the signal. Once the signal is represented in the spectral domain, *continuous spectral subtraction* (CSS) is applied. Continuous spectral subtraction is a variant of spectral subtraction, a simple method of removing noise by subtracting an estimate of the noise from the signal, described in section 4.1. The estimate is obtained by calculating the mean of the first frames of the signal, which do not yet contain speech information. In continuous spectral subtraction the mean is updated each frame.

In the final stage, the mel-frequency cepstrum of the signal is obtained by applying a 20 triangular bin filterbank, a log compression operation and a discrete cosine transform (DCT) to the signal. In the log cepstral domain, a *lifter* is applied to re-scale the cepstral coefficients to have similar magnitudes and *exact cepstral mean normalization* (E-CMN) is performed. Similar to spectral subtraction, cepstral mean normalization attempts to compensate for long term effects caused by different microphones and audio channels, by computing and the average of each cepstral coefficient and removing it from the signal. E-CMN differs from CMN in that two mean vectors are used to normalize the signal instead of one, one containing speech data and one containing non-speech data. CMN was discussed in section 4.1. Because of the custom nature of the acoustic analysis, a specially developed tool called *AcousticAnalysisC11kHz* is used to perform the acoustic feature vector extraction. The acoustic feature vectors are stored alongside the PCM files.

6.3.4 Pronunciation Dictionary

The pronunciation dictionary plays a role in determining the phonetic transcription of words uttered by speakers in the training corpus. It is a text file with an entry on each line, consisting of a headword followed by a phoneme sequence. The pronunciation dictionary used to train the Dutch acoustic models was compiled from two different sources: the transcription information belonging to the Dutch corpus and an off-the-shelf lexicon acquired from a company specialized in speech technology. The two sources differ in phoneme definition and transcription protocol, thus in order to merge them, they have to be transformed to a common format.

The phoneme definition is given in tables 6.2 and 6.3, which list the Dutch consonants and vowels respectively. Besides listing the phonemes in SAMPA format, tables 6.2 and 6.3 also list the *HTK subwords* associated with them. HTK subwords will be described in more detail in the next section.

Table 6.4 lists the total number of entries in the dictionary, along with the number of entries that have multiple pronunciation candidates associated with them.

6.3.5 HTK Subwords

The *HTK subwords* represent the actual hidden Markov models that will be trained by the HTK. As is apparent from table 6.2 and table 6.3, the set of phonemes is not mapped one-to-one to a set of subwords. There are several reasons for this. First, if a certain phoneme is uncommon, it is possible that there is limited or no speech data available to train a good model. In this case it is sensible to choose a subword to represent this phoneme that is also

Table 6.2 Dutch consonants.

Category	SAMPA	HTK	Example	Pronunciation
Plosive	p	p	*p*ak	p A k
	b	b	*b*ak	b A k
	t	t	*t*ak	t A k
	d	d	*d*ak	d A k
	k	k	*k*ap	k A p
	g	g	*g*oal	g o: l
Fricative	f	f	*f*el	f E l
	v	v	*v*el	v E l
	s	s	*s*ok	s O k
	z	z	*z*ak	z A k
	x	x	to*ch*	t O x
	G	x	*g*oed	x u t
	h	h	*h*ond	h O n t
	Z	ge	baga*ge*	b a: g a: Z @
	S	sj	*sj*aal	S a: l
Sonorant	m	m	*m*an	m A n
	n	n	*n*on	n O n
	N	ng	ba*ng*	b A N
	l	l	*l*am, ba*l*	l A m, b A l
	r	r	*r*and	r A n t
	w	w	*w*eer	w e: r
	j	j	*j*a, haa*i*	j a:, h a: j

Table 6.3 Dutch vowels.

Category	SAMPA	HTK	Example	Pronunciation
Checked	I	i	p*i*t	p I t
	E	e	p*e*t	p E t
	A	a	p*a*k	p A k
	O	o	p*o*t	p O t
	Y	u	p*u*t	p Y t
	@	sjwa	g*e*doe	G @ d u
Free	i	ie	p*ie*t	p i t
	y	uu	f*uu*t	f y t
	u	oe	v*oe*t	v u t
	a:	aa	p*aa*l	p a: l
	e:	ee	v*ee*l	v e: l
	2:	eu	b*eu*k	b 2: k
	o:	oo	b*oo*t	b o: t
	Ei	ei	st*ij*l, st*ei*l	s t Ei l
	9y	ui	h*ui*s	h 9y s
	Au	au	r*ou*w, r*au*w	r Au w
Diphthong	a:i	aai	dr*aai*	d r a:i
	o:i	ooi	m*ooi*	m o:i
	ui	oei	r*oei*	r ui
	iu	ieu	n*ieuw*	n iu
	yu	uw	d*uw*	d yu
	e:u	eeuw	sn*eeuw*	s n e:u
Marginal	E:	me	crème	k r E: m
	9:	meu	fr*eu*le	f r 9: l @
	O:	mo	z*o*ne	s O: n @
	e∼	e	v*in*	v e∼
	a∼	a	v*ent*	v a∼
	o∼	o	b*on*	b o∼
	9∼	u	br*un*, parf*um*	b r 9∼, p A r f 9∼

Table 6.4 Dictionary

dictionary revision 0706	
Total number of entries	16146
Total number of phonemes	51
Words with 1 variant	15672
Words with 2 variants	471
Words with 3 variants	3

used to represent a more common phoneme of similar sound. Table 6.3 shows that there are four marginal vowels that have no unique subword associated with them. Second, if two phonemes are very close in sound it makes sense to train just one model for both of them. Table 6.2 shows that /x/ and /G/ both have the same subword associated with them. Third, it is sometimes desirable to use two models to represent a certain phoneme. Diphthongs, for instance, can be split into a vowel and a semivowel (or glide) part, such as the Dutch phoneme /a:i/ which can be represented by the subword sequence `aa j`. Finally, it is possible that a phoneme is represented by different subwords in different circumstances. For example, different subwords can be chosen to represent a phoneme, depending on its position within a word or depending on the phonemes it is preceded or followed by.

The mechanism by which the mapping from SAMPA phonemes to HTK subwords takes place is in the form of *Pronunciation to Subword* (P2S) rules. The P2S rules are specified in a text file, which contains four sections: a list of the subwords, a list of the elements (i.e. SAMPA phonemes), a list of groups and a list of rules. The rules are written in a simple syntax. Each line is a rule, starting with an element sequence and ending with a subword sequence. Optionally, context parameters, increment width and rule priorities can be specified. Table 6.5 shows an excerpt of the P2S rule file. The complete file can be found in appendix B. Table 6.5 shows that /Y/ and /9∼/ are both mapped to subword `u`, as is /2:/, but only if followed by /r/.

In total 47 models are defined, 21 models representing consonants, 25 models representing vowels and a model representing pauses.

Table 6.5 Excerpt of P2S Rule file.

rule_begin				
p	*	:	1	p
x	*	:	1	x
G	*	:	1	x
Y	*	:	1	u
a:	*	:	1	aa
e:	*	:	1	ee
e:	r	:	1	i
2:	*	:	1	eu
2:	r	:	1	u
e∼	*	:	1	e
a∼	*	:	1	a
o∼	*	:	1	o
9∼	*	:	1	u

6.3.6 Time Alignment Information

In order to obtain an initial set of models *bootstrap data* is used. As mentioned in chapter 5, bootstrap data is speech audio data for which the phone boundaries have been indentified. The phone boundaries determine the start and end positions of individual phones within a speech data file. The HTK will determine initial values for the model parameters using this information.

For training the Dutch acoustic models, the bootstrap data is acquired by using pre-existing acoustic models from other languages closely related to Dutch. Using these foreign approximate HMMs, the HTK Viterbi recognizer will match speech files against a word-level network and output a transcription for each file. The transcriptions with associated time alignment information, are referred to as *label* files. The foreign approximate models used were mainly of German and English origin, although several French models were used as well.

Figure 6.5 illustrates the process of acquiring the time label files. The network files need to be created first. The concept of a word network was briefly described in chapter 5. The function of the network is to provide a grammar for the HTK tool *HVite* to perform the time alignment. Table 6.6 lists an example network for the Dutch words 'tulp' and 'kijk een'. The networks are created by a tool called *HTKNetworkMaker*. For each entry in the corpus list file, HTKNet-

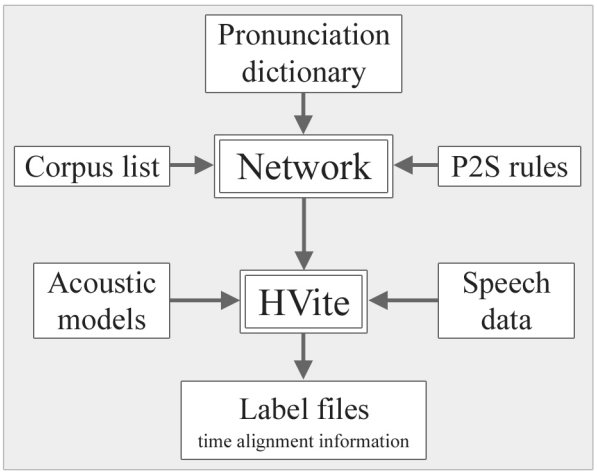

Figure 6.5 Time alignment process.

Table 6.6 HTK network files.

```
( pau
                ((k ei k))
[ pau ]         ((sjwa n))
pau )
```

```
( pau
                ((t u l sjwa p))
pau )
```

workMaker will transcribe the word label associated with the speech data file into a SAMPA phoneme sequence and, using the P2S rule mechanism, produce a network file with HTK subwords in the required HTK network syntax.

Once the network files have been created, *HVite* will process all the speech files in the training corpus and, using the associated network and the set of foreign approximate HMMs, create the HTK label files. Example label files belonging to the networks given in table 6.6 are listed in table 6.7. Once this process is complete all speech files in the training corpus have time aligned label files associated with them, thus the bootstrap data is the entire training corpus.

Table 6.7 HTK label files.

0	2793644	pau
2793644	3292509	k
3292509	5188196	ei
5188196	6185926	k
6185926	7383202	pau
7383202	8879797	sjwa
8879797	10276619	n
10276619	14566858	pau
0	2793644	pau
2793644	3392282	t
3392282	4190466	u
4190466	4589558	l
4589558	5387742	sjwa
5387742	6185926	p
6185926	10575938	pau

6.3.7 HMM Prototypes

In the final step of data preparation, the topology of the hidden Markov models is defined. All models share a similar topology, they have an entry state, an exit state and a number of states in between, depending on the phoneme the model represents. Transitions are from left to right only and skipping of states is not allowed. The output probability is determined by a single Gaussian density function.

The initial HMMs, called the prototypes, are created with a tool called *HMMPrototype*. HMMPrototype makes a separate file for each model and gives default values to the transition and output probability parameters. All means of the Gaussian density functions are set to 0 and the variances to 1.0. The number of states of each model is specified by a prototype definition file, which is listed in table 6.8. As is shown, most models have three emitting states. The models with more emitting states are the pause model, *pau*, and the models for the Dutch diphthongs, as these are longer sounds.

In the prototype HMM files only the basic structure of the HMMs is defined: the number of states, number of mixes (Gaussian mixture components) and number of streams (MFCC sets). The actual values of the parameters are determined in the training phase, which will be discussed in detail in the next section.

Table 6.8 HMM emitting states.

Number of emitting states											
pau	12	v	3	n	3	a	3	ee	3	oei	4
p	3	s	3	ng	3	o	3	eu	3	ieu	5
b	3	z	3	l	3	u	3	oo	3	uw	4
t	3	x	3	r	3	sjwa	3	ei	3	eeuw	5
d	3	h	3	w	3	ie	3	ui	3	me	3
k	3	ge	3	j	3	uu	3	au	3	meu	4
g	3	sj	3	i	3	oe	3	aai	5	mo	4
f	3	m	3	e	3	aa	3	ooi	5		

6.4 Training

This section will cover the training of the HMMs. Training takes place in two stages. First, initial estimates of the parameters of a single HMM are estimated by the HTK tools *HInit* and *HRest*. In the second stage, whole sets of models are re-estimated simultaneously by the Baum-Welch algorithm using the HTK tool *HERest*.

The training of the models is split between male and female gender. In total 47 models will be trained for each. Splitting the models between the genders is a form of *clustering*. Clustering will be described in more detail in chapter 7.

The training data consists of speakers from groups B, C, D and E; 60 speakers in total per gender. The corpus category **SABW** is used to train the phonemic based models, with 1308 utterances divided over the four speaker groups. Thus, a total of 19,620 utterances per gender is used in the training process.

6.4.1 Initial Models

The training process is illustrated in figure 6.6. Before applying the Baum-Welch algorithm to train the acoustic models, it is necessary to provide initial values for the model parameters. The choice of initial parameters is important as the HMMs are sensitive to them. To provide the initial parameters *HInit* and *HRest* are used. *HInit* uses the bootstrap data as described in the previous section, illustrated in figure 6.6. The label files provide phone boundary information which *HInit* uses to 'cut out' instances of each phone from parameterized speech data. Using a segmental k-means procedure *HInit* calculates initial means and variances for the models. In further iterations k-means is replaced by Viterbi alignment. *HInit* runs with an option to limit the maximum number of estimation cycles to 20. The output of *HInit* is input to *HRest* which further re-estimates the model parameters. *HRest* is similar to *HInit* in that it uses the time aligned label files to estimate the parameters of each model individually. Instead of the segmental k-means procedure however, *HRest* uses the Baum-Welch algorithm. The output of *HRest* is input to *HERest*

6.4.2 Embedded Re-estimation

The main HTK training tool is *HERest*, as discussed in chapter 5. Once initial values have been estimated for the model parameters, *HERest* performs an embedded re-estimation using the entire training set. This re-estimation is

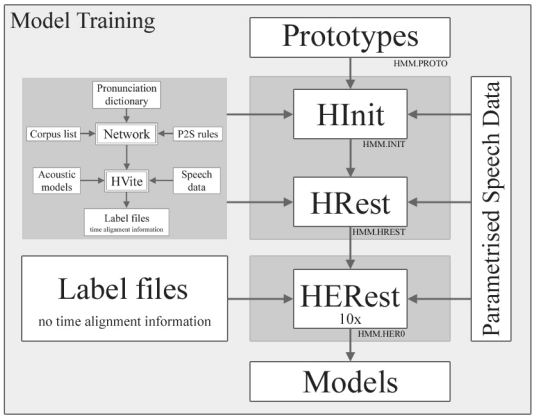

Figure 6.6 Model training overview.

performed by the Baum-Welch algorithm, using transcription label files without time alignment information. For each training utterance, the relevant phoneme models are concatenated into a composite model, for which *HERest* simultaneously updates all parameters by performing a standard Baum-Welch pass. This process is performed in two steps:

1. Each input data file contains training data which is processed and the accumulators for state occupation, state transition, means and variances are updated.

2. The accumulators are used to calculate new estimates for the HMM parameters.

HERest provides several optimizations to improve the speed of the training process. It is capable of pruning the transition and output probability matrices, thus also achieving a reduction in memory usage. It is also possible to operate *HERest* in parallel mode. When running in parallel, the training data is split into groups, which *HERest* can process individually. Accumulators for each group are stored into files. When all groups are processed, the accumulators are combined into a single set and used to calculate new estimates for the HMM parameters.

When an *HERest* process is completed, the set of acoustic models is fully trained. The training of models with *HERest* is performed a total of ten times, each trained set of models from one iteration forming the input for the next as initial models. The final HMMs are saved as simple text files, an example of which is included in appendix B. In the next section the evaluation of the acoustic models will be described.

6.5 Evaluation

As mentioned before, the Dutch language corpus contains utterances from a total of 75 speakers of each gender, 60 of which are used to train the Dutch

acoustic models. The final 15 speakers from each gender are used as *evaluation* speakers. The training is both *category closed* as well as *speaker closed*. That is, no category of utterances from the corpus used in training, is used in evaluation and no group of speakers used in training the acoustic models, is used in evaluating them.

The category used for training is the **SABW** category, which contains utterances spoken by speakers from groups B, C, D and E. The category used for evaluating the models is the **EVW** category, which contains utterances spoken by speakers from group A. The **EVW** category consists of 145 utterances.

Recognition of the evaluation speech data is performed by the HTK recognition tool *HVite*. The process is much similar to that of the creation of time label files as was described earlier. It is controlled by a recognition network, a dictionary and a set of HMMs. The process is illustrated in figure 6.7 and will be described in detail in the rest of this section.

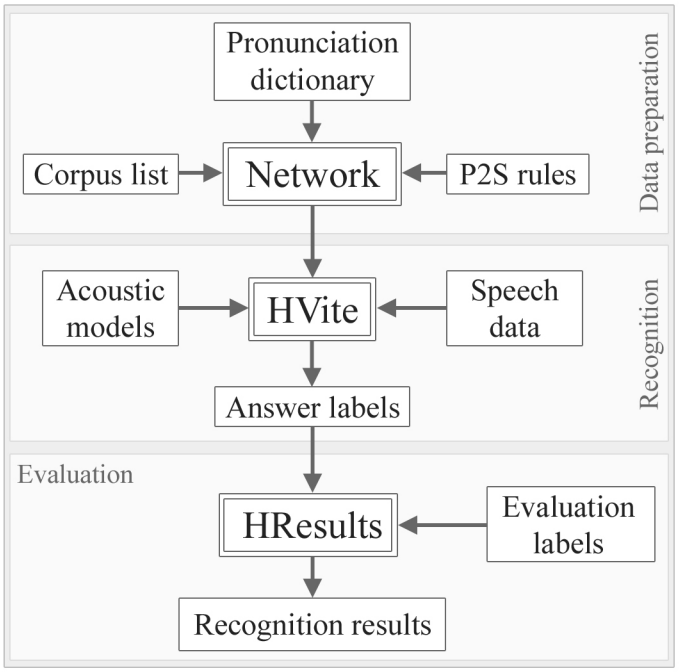

Figure 6.7 Overview of the evaluation process.

6.5.1 Data Preparation

The networks are created by a tool called *HTKNetworkMaker*, as mentioned in a previous section. For each entry in the corpus list file, HTKNetworkMaker will transcribe the word label associated with the speech data file into a SAMPA phoneme sequence and, using the P2S rule mechanism, produce a network file with HTK subwords in the required HTK network syntax. The recognition network is a simple word network as listed in table 6.9. If the network is large,

Table 6.9 HTK evaluation network.

```
( SIL
        ( aankoop ) |
        ( aardappel ) |
        ( plant ) |
        ( beker ) |
        ( aardbei ) |
        ( arbeider ) |
        ( aandacht ) |
        ( aardbeving ) |
        ( technisch ) |
        ( diploma ) |
        ... |
SIL )
```

HVite can perform *pruning*. Pruning is a method of reducing required computation. This is done by dismissing paths through the network with a log probability that falls below a certain threshold, called the *beam-width*. Besides the network, the evaluation labels have to be created. The evaluation labels are reference labels that, in order to calculate the model performance, are compared to the labels that *HVite* will output as recognition results. Sample evaluation labels are listed in table 6.10. The job of the decoder is to find those paths through the network which have the highest log probability. For all evaluation utterances *HVite* will output an answer label file containing the recognized word, its boundaries and its recognition probability. Example answer labels are listed in table 6.11.

Table 6.10 HTK evaluation label files.

0	0	aankoop
0	0	aardappel
0	0	plant

Table 6.11 HTK answer label files.

frames		**label**	**log probability**
3092963	10875257	aankoop	-4959.134766
3192736	10875257	aardappel	-5024.846191
2993190	8181386	plant	-3352.746582

6.5.2 Recognition Results

Once the evaluation data has been processed by the recognizer, the next step is to analyze the results. The analysis is performed by the HTK tool *HResults*. HResults compares the answer labels, as they are output by *HVite* with the evaluation labels, which contain transcriptions of the utterances as they should have been recognized. The comparison is performed using dynamic programming. Once HResults has found the optimal alignment between answer labels

and reference labels, the number of substitution errors (S), deletion errors (D) and insertion errors (I) can be calculated. The percentage of correctly recognized word is:

$$\text{Percent correct} = \frac{N - D - S}{N} \times 100\% \tag{6.1}$$

with N the total number of labels in the reference transcription. This measure ignores insertion errors. The the total model accuracy is defined as:

$$\text{Percent accuracy} = \frac{N - D - S - I}{N} \times 100\% \tag{6.2}$$

Every set of models obtained in the iterations of *HERest* is evaluated. The best models are often not found in the final iteration. This can be explained by the fact that the more *HERest* iterations are performed, the greater the risk of *over training* the acoustic models. The models are over trained when their output probabilities will match the training data with great accuracy but will match less well to any new speech data. Hence the final models are selected from the best iteration.

Table 6.12 shows the final results of the Dutch monophone phoneme models. Two sets of models were trained, mixed with different kinds of noise at different signal-to-noise ratios. These models were trained in version 1.2 of the VAMB environment. The results are the effect of various different training cycles all with different configurations of the training and model parameters.

The monophone models form a basic set of Dutch acoustic models. The next section will explain how these models are expanded to biphone models to take into account the effects of context dependency.

Table 6.12 Model_400 results.

Model			
Model Name	Model_400		
Created	01-Mar-04		
Training Information			
Category	SABW		
Noise Kind	Booth2		
SNR	20 dB		
VAD	n/a		
Results			
HVite	Female	Male	Average
	92.42	**90.37**	**91.40**
VV100	Female	Male	Average
	n/a	**n/a**	**n/a**

Model			
Model Name	Model_400		
Created	05-Mar-04		
Training Information			
Category	SABW		
Noise Kind	NAT_Car		
SNR	-5 dB		
VAD	n/a		
Results			
HVite	Female	Male	Average
	75.31	**71.11**	**73.21**
VV100	Female	Male	Average
	n/a	**n/a**	**n/a**

6.6 Context Dependent Models

Monophone models do not accurately reflect the nature of human speech. As was described in chapter 2, the realization of a phoneme depends greatly on its context, that is, preceding and following sounds. To better model human speech, it is necessary to include context into the design of acoustic models.

A common technique is to use triphone models. Triphone models contain a core phoneme and left and right contexts. Using triphones has several limitations. First, to consider all phonemes in all possible contexts requires a great many triphones to be created. This has severe effects on the memory consumption and computational speed of the recognizer. Second, to train triphone

models properly a considerable amount of training data is required. Usually in the design of triphones, the problem of data sparsity is taken care of by a variety of techniques including smoothing, parameter tying and model clustering.

In embedded environments, such as car navigation systems, the memory and computational power limits the total number of models a system can handle. A good alternative to using triphones to model context dependency, is the use of biphones.

The rest of this section will describe the process of creating and training a biphone phoneme acoustic model set for the Dutch language.

6.6.1 Biphone Models

To expand monophone models into biphones, a tool called *HMMChanger* is used. Essentially, *HMMChanger* combines two monophone models into one biphone model, for all possible combinations of monophones. This is done by splitting the monophone models. The exact mechanism by which this in done is proprietary to Asahi Kasei and will not be discussed. The naming convention of a biphone model is `leftMonophone_rightMonophone`. Figure 6.8 illustrates the creation of the biphone model `a_r` from the monophone models `a` and `r`. An example of a biphone transcription using HTK subwords is given in table 6.13.

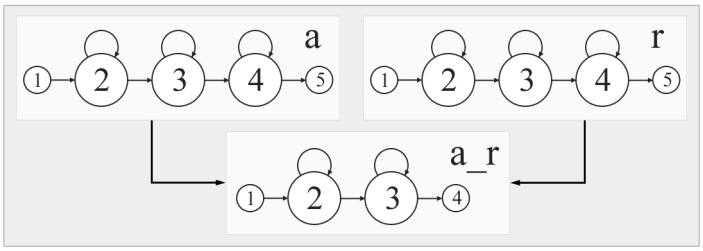

Figure 6.8 Creation of biphone models.

Table 6.13 Transcription of the Dutch word 'tulp' using biphones.

pau pau_t t t_u u u_l l l_sjwa sjwa sjwa_p p p_pau pau

6.6.2 Training

The biphone models are trained similar to the monophone models. As the initial models are already created using the monophones, only an embedded re-estimation is performed. In total, *HERest* runs for ten iterations, the output of each iteration forming the input of the next. As was mentioned before, *HERest* requires transcription information of all training data in the form of label files, but does not require time alignment information. The label files for the biphone training are obtained from the monophone label files using the tool *Uni2AceLabelChanger*, which changes the monophone labels to biphone labels

using a configuration file. The configuration file simply lists all biphone models and which two monophone models are used to create them. Table 6.14 lists two biphone label files belonging to the Dutch words 'tulp' and 'kijk een'.

The corpus training categories used to train the biphone models are the same as those used to train the monophone models. A distinction is also made between male and female models. In total 2255 models are trained for each gender. This number is the sum of all combinations of monophones (47^2) and the 47 *stable* parts (47). Subtracted from this is the model `pau_pau`, which is not trained.

Table 6.14 HTK biphone label files.

0	0	pau
0	0	pau_t
0	0	t
0	0	t_u
0	0	u
0	0	u_l
0	0	l
0	0	l_sjwa
0	0	sjwa
0	0	sjwa_p
0	0	p
0	0	p_pau
0	0	pau
0	0	pau
0	0	pau_k
0	0	k
0	0	k_ei
0	0	ei
0	0	ei_k
0	0	k
0	0	k_pau
0	0	pau
0	0	pau_sjwa
0	0	sjwa
0	0	sjwa_n
0	0	n
0	0	n_pau
0	0	pau

6.6.3 Recognition Results

The biphone models are evaluated in much the same way as the monophone models. The evaluation network, the evaluation labels and the noise conditions are all the same. All ten *HERest* iterations are evaluated. Table 6.15 shows the final results of the Dutch biphone phoneme models. Two sets of models were trained, mixed with different kinds of noise at different signal-to-noise ratios. The final results are the effect of various different training cycles, with different configurations of the training and model parameters.

The final section of this chapter covers the creation of monophone word models for the recognition of the Dutch digits and the Dutch alphabet.

Table 6.15 Biphone_400 results.

Model			
Model Name	Biphone_400		
Created	01-Mar-04		
Training Information			
Category	SABW		
Noise Kind	Booth2		
SNR	20 dB		
VAD	NORMAL		
Results			
HVite	Female	Male	Average
	96.77	**96.55**	**96.66**
VV100	Female	Male	Average
	97.69	**96.78**	**97.21**

Model			
Model Name	Biphone_400		
Created	05-Mar-04		
Training Information			
Category	SABW		
Noise Kind	NAT_Car		
SNR	-5 dB		
VAD	HIGHEST		
Results			
HVite	Female	Male	Average
	86.29	**82.57**	**84.43**
VV100	Female	Male	Average
	81.63	**79.79**	**80.60**

6.7 Word Models

In this section the creation of two more model sets will be described: models for recognition of Dutch digits and models for recognizing the Dutch alphabet. In contrast to previously created models, these models are word based. For each digit and each letter of the alphabet there is a unique model.

The process of training the word models is similar to that of training the phonetic based acoustic models. Although isolated word recognition is much simpler and less powerful than phonemic based recognition, it is well suited for command based environments, such as car navigation systems. The restricted vocabulary allows digits to be recognized with a high degree of accuracy.

6.7.1 Digit Models

The corpus training category to train the digit models is the **CCD** category. There are 26 utterances in this category, each utterance consists of five digits spoken consecutively. Speaker groups B, C, D and E contain the training speakers, 60 for each gender, making a total of 1560 training utterances per gender. Evaluation is done by 15 speakers from speaker group A, each speaking 26 utterances.

Table 6.16 lists the models that are trained as well as the model topology. The recognition network is given in table 6.17. It is a simple word loop network that specifies that any of the digits can be spoken, any number of times, with or without a pause in between. Recognition results are listed in table 6.18.

Table 6.16 Digit model topology.

Number of emitting states			
pau	12	vijf	17
nyl	12	zes	19
een	11	zeven	17
twee	15	acht	16
drie	11	negen	16
vier	15		

Table 6.17 Digit_500 evaluation network.

```
$digit = ( nyl | een | twee | drie | vier | vijf | zes | zeven | acht | negen );
( SIL [pau] $digit SIL )
```

Table 6.18 Digit_500 results.

Model			
Model Name	Digit_500		
Created	01-Mar-04		
Training Information			
Category	CCD		
Noise Kind	Booth2		
SNR	20 dB		
VAD	NORMAL		
Results			
HVite	Female	Male	Average
	95.36	**97.16**	**96.26**
VV100	Female	Male	Average
	93.31	**96.40**	**94.73**

Model			
Model Name	Digit_500		
Created	03-Mar-04		
Training Information			
Category	CCD		
Noise Kind	NAT_Car		
SNR	-5 dB		
VAD	HIGHEST		
Results			
HVite	Female	Male	Average
	60.58	**69.83**	**65.21**
VV100	Female	Male	Average
	42.53	**64.65**	**53.20**

6.7.2 Alphabet Models

The corpus training category to train the alphabet models is the **CSA** category. Each utterance is a letter from the Dutch alphabet. Speaker groups B, C, D and E contain the training speakers, 60 for each gender, each speaking 29 utterances. This makes a total of 1740 training utterances per gender. Evaluation is done by 15 speakers from speaker group A, each speaking 29 utterances.

Table 6.19 lists the models that are trained as well as the model topology. The Dutch language contains four different possibilities for expressing 'y'. These are IJ, i-grec, griekse-y and ypsilon. Recognition results are listed in table 6.20.

Table 6.19 Alphabet model topology.

Number of emitting states												
pau	12	F	13	L	10	R	9	X	16
A	9	G	13	M	10	S	14	IJ	10
B	11	H	11	N	10	T	10	Z	15
C	13	I	7	O	9	U	7	i-grec	18
D	11	J	12	P	9	V	13	griekse-y	25
E	9	K	10	Q	8	W	12	ypsilon	20

Table 6.20 Alphabet_120 results.

Model			
Model Name	Alphabet_120		
Created	27-Feb-04		
Training Information			
Category	CSA		
Noise Kind	Booth2		
SNR	20 dB		
VAD	NORMAL		
Results			
HVite	Female	Male	Average
	93.33	**94.94**	**94.14**
VV100	Female	Male	Average
	93.33	**93.56**	**93.10**

Model			
Model Name	Alphabet_120		
Created	03-Mar-04		
Training Information			
Category	CSA		
Noise Kind	NAT_Car		
SNR	-5 dB		
VAD	NORMAL		
Results			
HVite	Female	Male	Average
	82.76	**80.00**	**81.38**
VV100	Female	Male	Average
	80.95	**83.77**	**82.10**

Chapter 7

Advanced Topics

In this chapter a number of techniques will be discussed related to the optimization of the acoustic models developed for the Dutch language. These techniques are: use of acoustic visualization, adding Gaussian mixture components and reduction of the number of biphones.

7.1 Acoustic Visualization

Demand for practical application of HMM-based speaker-independent automatic speech recognition continues to grow each year. Very satisfactory results can be achieved with the current generation of speech recognition software, yet high results are usually restricted to a relatively small group of speakers, that speak in an ideal manner, that is, close to the speakers of the development set. Furthermore, variation in ambient noise can severely affect recognition performance. In section 4.1 variability in the speech signal is discussed in detail.

Completely speaker-independent speech recognition is thus yet to be achieved. There are several speaker-adaptation methods that enable the recognition rate to increase given some sample data by a particular speaker. The most common techniques developed for this purpose, based on MAP estimation and MLLR were discussed in section 4.2. It is, however, often difficult to obtain a sufficient amount of voice samples. Users find it troublesome to spend a lot of time 'training' a system to recognize their voice and are often unmotivated for this task. A typical user expects a speech recognition system to provide high recognition from the moment of first activation.

An alternative approach to adaptation is the use of a *library* of acoustic models [21] corresponding to all possible variability in the speech signal as described in section 4.1. A speech recognition system could potentially select the most appropriate acoustic models from the library depending on the circumstances. The acoustic models in the library can compensate for differences among human speakers, but can also contain models for varying environmental circumstances. In car navigation systems different acoustic models can correspond with different engine types and other kinds of noise.

The development of acoustic models is not a trivial task and developers are scarce. Developing acoustic models requires a high level of expertise and advanced knowledge of hidden Markov models. This section introduces a visu-

alization system that provides a two-dimensional visualization of the acoustic space, which facilitates the construction of advanced acoustic model libraries by developers that can use visual cues for classification of acoustic models.

7.1.1 COSMOS

In order to increase the accuracy of acoustic models, it is essential to comprehend the configuration of the acoustic space formed by the voice and noise signals as processed by a speech recognition system. This can be achieved by visualization of multidimensional acoustic information mapped onto a space of lower order, using a process referred to as *Multidimensional Scaling* (MDS). A visual mapping onto a two-dimensional space using the MDS method, typically involves approaches based on principal component analysis, discriminative analysis and others. All techniques, however, perform two-dimensional projections of multidimensional vectors and are thus not suitable for projection of multidimensional Gaussian distributions, such as those used in acoustic models.

A technique developed by Shozokai [21] at Asahi Kasei, does allow for HMMs to be mapped onto a two-dimensional space. This is a nonlinear projection technique based on the Sammon method [20]. In general, an acoustic model set is regarded as consisting of multiple acoustic models. The distance $D(i,j)$ between acoustic model set i and j is defined as follows:

$$D(i,j) \equiv \frac{1}{K} \sum_{k=1}^{K} d(i,j,k) \cdot w(k) \tag{7.1}$$

with K the total number of acoustic models, $d(i,j,k)$ the mutual distance between acoustic model k within model set i and the acoustic model k within model set j and $w(k)$ the occurrence frequency of acoustic model k. The resulting representation is referred to as the *Acoustic Space Map Of Sound* (COSMOS). An acoustic model set projected onto the COSMOS is referred to as a *STAR*. Using the Euclidean distance of mean vectors normalized by variance vectors, $d(i,j,k)$ can be expressed as:

$$d(i,j,k) = \frac{1}{S(k)} \sum_{s=0}^{S(k)-1} \frac{1}{L} \sum_{l=0}^{L-1} \frac{\big(\mu(i,k,s,l) - \mu(j,k,s,l)\big)^2}{\sigma(i,k,sl) \cdot \sigma(j,k,s,l)} \tag{7.2}$$

with $\mu(i,k,s,l)$ and $\sigma(i,k,s,l)^2$ the mean and variance vectors of dimension l for state s of acoustic model k within acoustic model set i, $S(k)$ the number of states of acoustic model k and L the number of dimensions of the acoustic models.

The plotting of the two dimensional position of each acoustic model results in a COSMOS map. Several COSMOS maps are illustrated in the next subsection.

7.1.2 COSMOS Maps

In figure 7.1 the acoustic space of the Dutch digit category *common connected digit* (CCD) is plotted on a COSMOS map. Each STAR corresponds to a particular speaker. There are 75 female speakers, represented by red dots, and 75 male speakers, represented by blue squares. What is immediately apparent, is that there are two very distinct clusters, corresponding to each gender. An

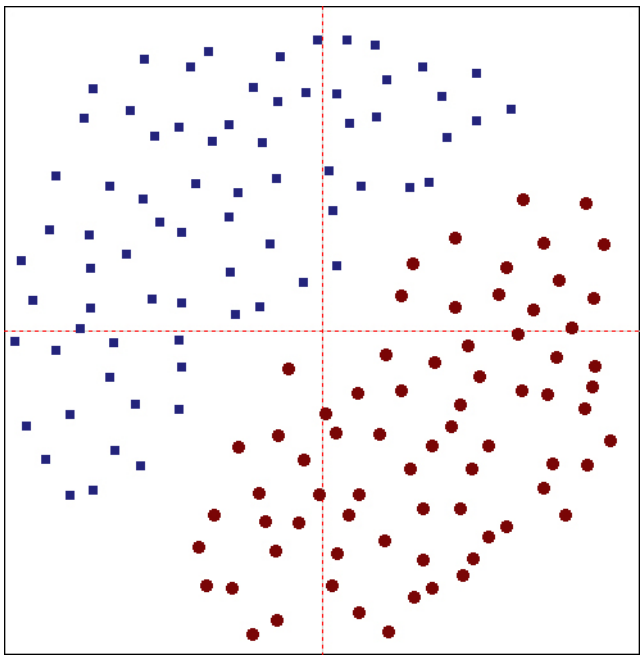

Figure 7.1 Cosmos CCD female and male.

obvious separation in two dimensional space also means a separation in the multi dimensional space. This supports the fact that a model set is created for each gender, as was discussed in chapter 6.

The information contained within the COSMOS map of figure 7.1 can also be used to decide which of the speakers will be designated as training and which as evaluation speakers. The COSMOS maps show the distribution of all the speakers in two dimensional space and thus training and evaluation speakers can be chosen so as to be properly distributed across the map.

Using COSMOS maps different properties of acoustic models can be investigated:

- **Gender**. In accordance with figure 7.1, plotting male and female acoustic model in the same map usually shows two distinct clusters, one for each gender.

- **Signal-to-noise ratio**. Voice data contaminated with noise at varying signal-to-noise ratio (SNR) shows up in different cluster on the COSMOS map.

- **Task**. Acoustic models created to recognize digits will show up on a different part of the map if plotted together with models created for another task, such as recognizing the alphabet.

- **Speaking style**. Whispering or speaking at higher pitch, for example, leads to distinctive clusters on a COSMOS map.

7.2 Multiple Mixture Components

The acoustic models developed for the Asahi Kasei VORERO middleware platform, use a single Gaussian density function as output probability function, as was discussed in 1. This restriction is based on the computational expense related to the more complex multivariate Gaussian mixture density functions. The HTK software, however, is not limited to the single Gaussian as output probability function, and it is interesting to see how the acoustic models perform using a different number of mixture components.

The multivariate Gaussian mixture output probability function is described as follows. Given M Gaussian mixture density functions:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} c_{jk}\mathcal{N}(\mathbf{x}, \mu_{jk}, \mathbf{\Sigma}_{jk}) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{x}) \tag{7.3}$$

with $\mathcal{N}(\mathbf{x}, \mu_{ij}, \mathbf{\Sigma}_{jk})$, or $b_{jk}(\mathbf{x})$, a single Gaussian density function with mean vector μ_{jk} and covariance matrix $\mathbf{\Sigma}_{jk}$ for state j, M the number of mixture-components and c_{jk} the weight of the k^{th} mixture component, which satisfies:

$$\sum_{k=1}^{M} c_{jk} = 1 \tag{7.4}$$

Figure 7.2 illustrates a multivariate Gaussian mixture probability density function. As is apparent, the multivariate Gaussian mixture probability function is essentially a superposition of individual Gaussian density functions, each with an own mean and variance.

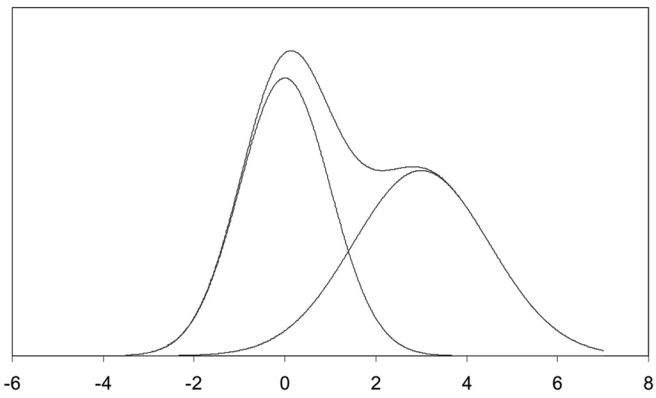

Figure 7.2 Multivariate Gaussian mixture density function.

Results of experiments carried out with an increased number of mixtures are listed in table 7.1. In model set *Model_610*, the number of mixture components per stream has been increased to two, for all models. In model set *Model_611*, all streams of all models have three mixture components. Both model sets share the same training conditions as the models sets described in chapter 6. If compared to the results of section 6.5, a substantial increase in performance is noticeable.

Table 7.1 Model_61x results.

Model			
Model Name	Model_610		
Created	26-Jul-04		
Training Information			
Category	SABW		
Noise Kind	Booth2		
SNR	20 dB		
VAD	n/a		
Results			
HVite	Female	Male	Average
	93.53	**92.37**	**92.95**
VV100	Female	Male	Average
	n/a	**n/a**	**n/a**

Model			
Model Name	Model_611		
Created	26-Jul-04		
Training Information			
Category	SABW		
Noise Kind	Booth2		
SNR	20 dB		
VAD	n/a		
Results			
HVite	Female	Male	Average
	94.13	**93.16**	**93.65**
VV100	Female	Male	Average
	n/a	**n/a**	**n/a**

7.3 Biphone Reduction

When designing for an embedded platform, memory consumption and processing speed are of crucial importance. Considering this, it is important to minimize the total number of acoustic models that are to be used, as less acoustic models require less calculations for the decoder and less memory is consumed.

The total number of biphones in the Dutch phoneme based model set, as described in section 6.6, is 2255, for each gender. This number is the sum of all combinations of monophones (47^2) and 47 *stable parts* (47). Subtracted from this is the model `pau_pau`, which is not trained.

Several methods exist to handle the issue of reducing the total number of models. A possible method is based on analysis of the dictionary. The dictionary file used in training the acoustic models provides a number of insights into what models can possibly be excluded from the model set. Depending on the language, there are always certain combinations of phonemes that will never occur. Table 7.2 lists a number of biphones, constructed from monophones, that are invalid for the Dutch language. To find similar invalid combinations,

Table 7.2 Invalid biphones.

HTK Subword	Left SAMPA	Right SAMPA
`a_a`	A	A
`aai_ieu`	a:i	iu
`h_ng`	h	N
`p_ng`	p	N
`b_v`	b	v

the most obvious method is to analyze the dictionary to see what combinations occur, and thus determine which ones do not. A total of 1170 biphones, combinations of phonemes, can be extracted from the dictionary. This, however, would not constitute all the valid possibilities. Consider a word ending on a certain phone concatenated with a word beginning with a certain phone. the combination of these two phones would also be a valid biphone. According to the words listed in the dictionary, there are 42 phonemes occurring in *head* position, and 41 phonemes occurring in *tail* position. Thus, a total of 1722 tail_head combinations are found. The union of the sets of combinations previously calculated yield a total of 1824 valid biphones, and 431 invalid biphones. The invalid biphones, however, contain all the combinations with the

pau model in both head and tail position, as well as all the models representing the *stable parts*, described in section 6.6. Correcting for these, the dictionary analysis method allows a reduction of 292 biphones, or 13%. Table 7.3 lists the calculation described above.

Merely relying on the information contained within the dictionary allows only modest reduction in the number of acoustic models.

Table 7.3 Invalid biphone count.

Total number of biphones:	2255
Biphones extracted from dictionary:	1170
Phonemes occurring at head position:	42
Phonemes occurring at tail position:	41
Tail_head combinations:	1722
Union of the sets:	1824
Missing biphones:	431
Correction for pau model:	292
Total reduction:	13 %

Chapter 8

Conclusion

In this final chapter an overview will be presented of how the research objectives stated in the introduction of this thesis have been addressed. Some future developments of speech recognition will also be discussed.

8.1 Research Objectives

The main research objective, *the addition of support for the Dutch language to VORERO*, was successfully realized. The pronunciation dictionary and the Dutch acoustic models were added to the VORERO SDK 6.0, which was released in the spring of 2004. In this section the completion of the individual tasks leading to the realization of the main research objective are discussed.

Understanding of current speech recognition technology, by studying relevant literature.

Speech recognition is a popular research topic. A lot of research on the subject is done around the world, both in private companies, as well as in universities and publicly funded programs. International conferences related to speech recognition technology are held regularly and there are several journals that publish new speech recognition developments on a monthly basis. In studying contemporary publications on speech recognition, two things can be remarked. First, recent developments are often focussed on a very small part of the speech recognition problem, making it difficult to place in a bigger context. Second, current speech recognition systems are based on proven research carried out over a number of years. Cutting-edge research has yet to prove its worth.

Chapter 4 contains most of the results of studying relevant speech recognition literature. The approach taken was to see what 'key challenges' face modern speech recognition research. One of the main challenges is robustness. As was described, robustness relates to the design of a system capable of understanding anyone's speech in all possible environments. Two of the environmental compensation techniques, described in section 4.1, have also been implemented in the VORERO system. These are *spectral subtraction* and *cepstral mean normalization*. They are related to the acoustic analysis described in section 6.3. With regard to environmental model adaption, the VORERO acoustic models

have been trained to include environmental noise as was described in section 6.3.

Understanding of the mathematical principles involved in stochastic speech recognition using hidden Markov model theory.

There is much mathematics involved in stochastic speech recognition. Basic mathematical theory is discussed in chapter 3. Acoustic analysis was discussed and the hidden Markov model introduced. Three of the main issues related to hidden Markov models were addressed: the evaluation problem, the decoding problem and the learning problem. Also in chapter 3 the application of HMMs to modeling human speech was discussed.

By understanding the mathematics of speech recognition, better acoustic models can be trained. Well-founded decisions can be made regarding model topology and other model parameters and valuable insight is gained for correct analysis of training results.

Study of the Dutch phoneme set and Dutch pronunciation rules.

The concept of a phoneme was introduced in chapter 2 and the complete Dutch phoneme set was described. Chapter 2 contains a systematic investigation into the mechanics of human speech. This information is relevant to the design of the acoustic models as described in section 6.3. The acoustic models reflect the phonemes, but not necessarily on a one-to-one basis.

The study of production of the Dutch speech sounds has allowed well-founded decisions to be made related to the design of the acoustic models.

Design of the Dutch acoustic models using the Hidden Markov Toolkit (HTK).

Chapters 5 and chapter 6 are related to the development of the Dutch acoustic models. In chapter 5 the HTK was described. The HTK has been found to be a flexible environment for training acoustic models. Chapter 6 is the core chapter of this thesis work. All steps required to training acoustic models using HTK were discussed in detail. Section 6.3 is particularly related to the *design* of the Dutch acoustic models and contains all the design decisions that were made.

Design of the acoustic models, represented by HTK subwords, includes finding the right amount of subwords to train, and establishing a correct mapping between the subwords and the Dutch phoneme set. The mapping between HTK subwords and the phonemes is essential to good speech recognition performance and many different mapping configurations were tried.

Training of the Dutch acoustic models using the HTK.

The training of the models was discussed in section 6.4. With all the required data in place, the actual training requires very little supervision. Setting up a training is, however, a nontrivial task. Model training is controlled by many parameters and finding the proper values for these parameters is a time-consuming process. Also, the development environment (VAMB) and the HTK are both

poorly documented and provide very little feedback in case of error. Much difficulty was experienced in detecting and correcting anomalies in the training process.

Four sets of acoustic models were trained: a monophone phoneme set, a biphone phoneme set, a word digit model set and a word alphabet model set.

Evaluation and optimization of Dutch speech recognition.

Once the Dutch acoustic models had been trained their performance needed to be evaluated. This process was discussed in section 6.5. The speech data not used in the training process was used for the evaluation. The recognition results given in chapter 6 were obtained using HTK and version 1.2 of the VAMB development environment.

The first model set trained and evaluated was not the final release set. The tuning of certain model parameters allowed a higher performance to be obtained. The number of emitting states in a model is one of such parameters. In chapter 7 model optimization was also discussed. The COSMOS visualization tool provides a method to visualize the acoustic model space, allowing better models to be trained.

Design of the Dutch pronunciation dictionary.

Design of the pronunciation dictionary was discussed in section 6.3. It was succesfully assembled from two different sources and converted to VORERO format.

Addition of the Dutch language to the VORERO SDK 6.0.

The Dutch acoustic model that were designed, trained and optimized as discussed in chapter 6 were successfully added to the VORERO system.

8.2 Future Developments

Speech recognition technology today is still a niche market and likely to remain that way for several more years [3]. Although speech recognition applications are pre-installed on modern PCs, the PC will not be the driving force behind adoption of speech recognition technology. Current trends indicate that next-generation mobile devices, such as tablet PCs, PDAs and cell phones, are being fitted with speech recognition capabilities. Car navigation systems are also ideally suited for speech recognition systems as the driving environment limits the use of hands for system interaction.

Some promising areas of current speech recognition research include the addition of *semantic knowledge* to recognition systems. Today's systems are designed with the goal to recognize a user's speech, though essentially a user doesn't want his speech to be recognized, he wants to perform a certain task. Adding semantic knowledge to a system means making the system understand the actual speech being recognized and will make computer systems significantly more efficient.

It has been found that current stochastic speech recognition architectures have a limit to minimal word-error rates. This means that in order to improve

accuracy beyond a certain point, systems might need to be augmented by other data. This is referred to as *multimodality*. Examples of multimodality are including eye- and lip movement data into current system design [9]. Many research efforts are currently focused on this.

Appendix A

Speech Recognition Research

A.1 History of Speech Recognition

In order to properly comprehend current global speech recognition research, this section will attempt to place the ongoing effort in a historical context.

A.1.1 The Early Years

Arguably the first machine to respond to the human voice was a toy dog with the name *Radio Rex* [3]. Manufactured sometime in the 1920s, Rex was designed to respond to its name. On picking up enough acoustic energy around 500 Hz, an electromagnetic bridge would break an electric circuit causing a spring to release Rex from the cage he was housed in. Rex's major weakness, his inability to properly discern between his name being spoken and similar sounding utterances, continues to plague speech recognition researchers today.

Throughout the 1930s and 1940s very limited advances were made in the field of speech recognition research, though there were modest improvements in voice compression and speech analysis techniques. It was in the 1950s that the world saw the birth of the first computerized word recognition system. Developed at AT&T's Bell Laboratory in 1952, the system could recognize an input of digits between zero and nine, spoken by a single speaker with significantly long pauses in between. Speech research in this period also introduced the use of phonemes as basic linguistic units in recognition systems, leading to a systems developed at MIT in 1959 capable of recognizing vowel sounds with a 93% accuracy rate.

Research in the 1960s remained primarily focused on acoustic models. In 1966 MIT improved their system to be able to cope with a 50 word vocabulary instead of mere vowels. Speech recognition received a large amount of public attention in 1968 with the release of Stanley Kubrick's classic space saga *2001: A Space Odyssey*, featuring the intelligent HAL 9000 computer system. The movie unfortunately set unrealistically high expectations for speech recognition and understanding.

A.1.2 DARPA Funds Speech Recognition Research

In the 1970s two things occurred that significantly advanced the field of speech recognition research. First, Hidden Markov Model (HMM) theory was introduced to model speech sounds. Second, the U.S. Department of Defense's Advanced Research Projects Agency (DARPA) decides to fund a five-year study of speech recognition.

HMM theory was developed in the late 1960s by L.E. Baum and J.A. Eagon working at the Institute for Defense Analyses (IDA) [16]. In the early 1970s Jim and Janet Baker, researchers at Carnegie Mellon University (CMU) applied HMM theory to continuous speech recognition. The Hidden Markov Model is a sophisticated stochastic technique that uses probability distributions to model speech sounds and has grown to be the dominant acoustic modeling approach in speech recognition research.

Since the 1940s the U.S. Department of Defense had pursued an active interest in human language technology, its primary goal being to develop a system capable of automatically decoding and translating Russian messages. All attempts to this end turned out as failures, but with speech technology at peak public appreciation, in 1971 DARPA established the Speech Understanding Research (SUR) program in order to develop a computer system that could understand continuous speech. The SUR Advisory Board specified the system should be able to recognize normally spoken English in a quiet environment with a 1000 word vocabulary, reasonable response times and an error rate of less than 10%. The main contractors of the SUR program were Carnegie Mellon University (CMU), Stanford Research Institute (SRI), MIT's Lincoln Laboratory, Systems Development Corporation (SDC) and Bolt, Beranek and Newman (BBN).

Systems developed at CMU during this period included HEARSAY-I and DRAGON, and later HEARSAY-II and HARPY of which the latter was the most impressive at the time, being able to recognize complete sentences consisting of a limited range of grammar structures. HARPY required around 50 state of the art computers to performs its calculations and could recognize 1011 words with a 95% accuracy rate. The main systems developed at BNN were SPEECHLIS and HWIM (Hear What I Mean).

In 1976 the HEARSAY-I, HARPY and BNN's HWIM were evaluated by DARPA. Other systems, including one co-developed by SRI and SDC, were not evaluated. CMU's HARPY outperformed the other systems, though because the SUR board had never fully specified its evaluation criteria, some researchers disputed the test results. This led to a great deal of controversy and eventually DARPA was forced to cancel funding for the SUR program and a five year follow-up study.

Funding for speech recognition research was resumed in 1984 as part of DARPA's Strategic Computing Program. As well as many of the original contractors several private companies took part including IBM and Dragon Systems, founded in 1982 by CMU researchers Jim and Janet Baker. In order to minimize testing controversies, full system evaluation standards and guidelines were laid down in advance by DARPA and the National Institute of Standards and Technology (NIST).

In 1989 CMU's SPHINX system wins the DARPA evaluation. From 1990 on private companies started determining the landscape of speech recognition research.

A.2 Timeline of Speech Recognition Research

1920	Commercial toy dog *Radio Rex* responds to 500Hz sounds which trigger an electromagnetic circuit to release him from his cage.
1936	AT&T's Bell Labs produces the first electronic speech synthesizer called the Voder (Dudley, Riesz and Watkins). This machine is demonstrated at the 1939 World Fair by experts that use a keyboard and foot pedals to play the machine and emit speech.
1952	AT&T Bell Laboratory develops a crude discrete, speaker dependent single digit recognition system. The world's first computerized 'word' recognition system.
1959	MIT develops a system that successfully identifies vowel sounds with 93% accuracy.
1966	New MIT system is able to cope with a 50 word vocabulary.
1967	L.E Baum and J.A. Eagon develop Hidden Markov Model theory at the Institute for Defense Analyses(IDA).
1968	The world-popular science fiction movie *2001: A Space Odyssey* introduces the idea of speech recognition with the space ship computer, HAL and set high public expectations.
1969	John Pierce of Bell Labs says automatic speech recognition will be infeasible for several decades because artificial intelligence is a prerequisite.
Early 1970's	Jim and Janet Baker apply HMM theory to speech recognition research at Carnegie Mellon University(CMU).
1971	U.S. Department of Defense's Advanced Research Projects Agency(DARPA) funds the Speech Understanding Research(SUR) program, a five-year study to determine the feasibility of automatic speech recognition.
1971 - 1973	HEARSAY-I, DRAGON, HEARSAY-II and HARPY systems are developed at Carnegie Mellon University. SPEECHLIS and HWIM are developed at Bolt, Beranek and Newman.
1976	DARPA evaluates speech recognition systems designed for the SUR program. Funding is canceled because of controversy over testing results.
1978	Texas Instruments introduces the popular toy *Speak and Spell* which used a speech chip leading to huge strides in the development of more human-like sounding synthesized speech.

1982	Dragon Systems is founded by Carnegie Mellon University researchers Jim and Janet Baker.
1984	DARPA resumes funding of speech recognition research as part of the Strategic Computing Program.
1984	SpeechWorks, a company providing state-of-the-art automated speech recognition telephony solutions is founded.
Early 1990's	Japan announces a *fifth generation* computing project. The effort is primarily intended for Japan to catch up with US software production, considered to be more advanced at the time. The five year effort included an attempt at machine translation and extensive speech recognition research.
1994	Dragon Systems releases DragonDictate, the first software-only commercial automated transcription product for the personal computer.
1996	The consumer company, Charles Schwab becomes the first company to implement a speech recognition system for its customer interface.
1997	Dragon Systems releases "Naturally Speaking," the first continuous speech dictation software.
1999	Japan commences government sponsored five year speech research project.
2002	TellMe supplies the first global voice portal, and later that year, NetByTel launched the first voice enabler. This enabled users to fill out a web-based data form over the phone.
2003	Dragon Naturally Speaking version 7.0 released.

Appendix B

Training Files

In this appendix some of the tools and configuration files are listed that are required by the HTK or VAMB environment to train acoustic models.

B.1 VAMB Tools

B.1.1 MakeModelWizard

Figure B.1 is a screenshot of the VAMB *MakeModelWizard*.

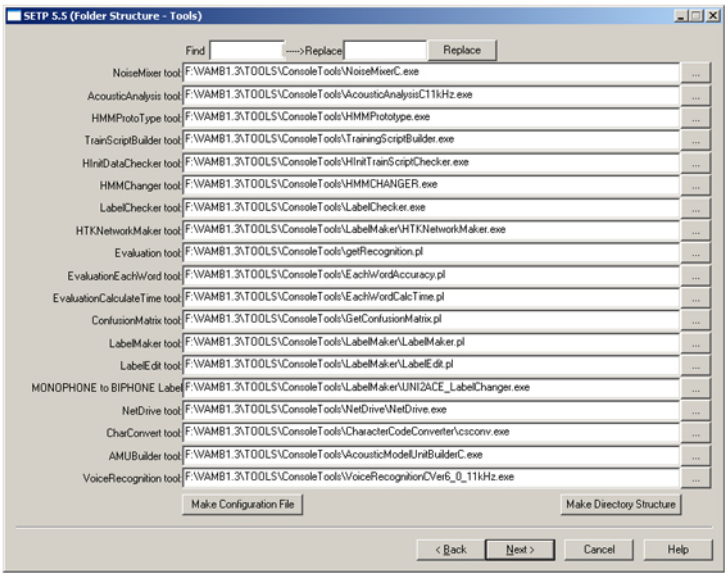

Figure B.1 VAMB MakeModelWizard

B.1.2 Perl Scripts

The following Perl scripts control the process of creating acoustic models using HTK.

1. **00_MakeTrainingDir.pl**
   Creates a folder structure in which all files created in the training process will be stored.
2. **005_CopyCleanData.pl**
   Copies the original speech data files from a backup server to the local workstation.
3. **01_NoiseMixTrainData.pl**
   Mixes the training speech data files with the noise specified in the configuration file.
4. **02_AcousticAnalysisTrainData.pl**
   Performs the acoustic analysis on the training data. Outputs feature vector files.
5. **023_PrepareForTimeLabelMaker.pl**
   Prepares network files and the dictionary needed to create time label files.
6. **025_TimeLabelMaker.pl**
   Creates time label files.
7. **03_MakeHMMPrototype.pl**
   Create the initial HMM model files.
8. **04_MakeHInitTrainingScript.pl**
   Creates a list of all the speech data files that will be used to run *HInit*.
9. **05_HInitTrainingScriptCheck.pl**
   Performs a check of all files required in the training process.
10. **06_HInit.pl**
    Runs the HTK *HInit* tool.
11. **07_HRest.pl**
    Runs the HTK *HRest* tool.
12. **08_FromHRESTtoHER0.pl**
    Copies the output from *HRest* to a new folder.
13. **085_AllLabelMaker.pl**
    Creates the label files used to train biphones.
14. **09_LabelCheck.pl**
    Performs a check of the label files.
15. **10_MakeHERestTrainingScript.pl**
    Creates a list of all speech data files that will be used to run *HERest*.
16. **11_HERest.pl**
    Runs the HTK *HERest* tool.

17. **12_NoiseMixEVData.pl**
    Mixes the evaluation speech data files with the noise specified in the configuration file.

18. **13_AcousticAnalysisEVData.pl** Performs the acoustic analysis on the evaluation data. Outputs feature vector files.

19. **14_PrepareForEvaluation.pl**
    Create reference label files.

20. **145_MakeEvaluationScript.pl**
    Creates a list of all speech data files that will be used in the evaluation of the acoustic models.

21. **15_Evaluation.pl**
    Performs the evaluation of the acoustic models using the HTK *HVite* tool.

B.2 HMM List

pau
a
aa
aai
au
b
d
e
ee
eeuw
ei
eu
f
g
ge
h
i
ie
ieu
j
k
l
m
me
meu
mo
n
ng
o
oe
oei
oo
ooi
p
r
s
sj
sjwa
t
u
ui
uu
uw
v
w
x
z

B.3 HMM Prototype List

```
pau     12
a       3
aa      3
aai     5
au      3
b       3
d       3
e       3
ee      3
eeuw    5
ei      3
eu      3
f       3
g       3
ge      3
h       3
i       3
ie      3
ieu     5
j       3
k       3
l       3
m       3
me      3
meu     4
mo      4
n       3
ng      3
o       3
oe      3
oei     4
oo      3
ooi     5
p       3
r       3
s       3
sj      3
sjwa    3
t       3
u       3
ui      3
uu      3
uw      4
v       3
w       3
x       3
z       3
```

B.4 P2S Rules

```
//=========================================
// RULE STRUCTURE DEFINITION
//=========================================
RULE_INFO_BEGIN
2       0
RULE_INFO_END
//=========================================
// SUBWORD DEFINITION
//=========================================
SUBWORD_BEGIN
pau
p
b
t
d
k
g
f
v
s
z
x
h
ge
sj
m
n
ng
l
r
w
j
i
e
a
o
u
sjwa
ie
uu
oe
aa
ee
eu
oo
ei
ui
au
aai
ooi
oei
ieu
uw
eeuw
me
meu
mo
SUBWORD_END
//=========================================
// PHONEME ELEMENTS DEFINITION
//=========================================
ELEMENT_BEGIN
p
b
t
d
k
g
f
v
s
z
```

```
x
G
h
Z
S
m
n
N
l
r
w
j
I
E
A
O
Y
@
i
y
u
a:
e:
2:
o:
Ei
9y
Au
a:i
o:i
ui
iu
yu
e:u
E:
9:
O:
e~
a~
o~
9~
ELEMENT_END
//=======================================
// GROUP NAME DEFINITION
//=======================================
GROUP_BEGIN
p       p
b       b
t       t
d       d
k       k
g       g
f       f
v       v
s       s
z       z
x       x
G       G
h       h
Z       Z
S       S
m       m
n       n
N       N
l       l
r       r
w       w
j       j
I       I
E       E
A       A
O       O
Y       Y
@       @
```

```
i       i
y       y
u       u
a:      a:
e:      e:
2:      2:
o:      o:
Ei      Ei
9y      9y
Au      Au
a:i     a:i
o:i     o:i
ui      ui
iu      iu
yu      yu
e:u     e:u
E:      E:
9:      9:
O:      O:
e~      e~
a~      a~
o~      o~
9~      9~
GROUP_END
//=========================================
// RULE DEFINITION
//=========================================
RULE_BEGIN
p       *       :       1       1       p
b       *       :       1       1       b
t       *       :       1       1       t
d       *       :       1       1       d
k       *       :       1       1       k
g       *       :       1       1       g
f       *       :       1       1       f
v       *       :       1       1       v
s       *       :       1       1       s
z       *       :       1       1       z
x       *       :       1       1       x
G       *       :       1       1       x
h       *       :       1       1       h
Z       *       :       1       1       ge
S       *       :       1       1       sj
m       *       :       1       1       m
n       *       :       1       1       n
N       *       :       1       1       ng
l       *       :       1       1       l
r       *       :       1       1       r
w       *       :       1       1       w
j       *       :       1       1       j
I       *       :       1       1       i
E       *       :       1       1       e
A       *       :       1       1       a
O       *       :       1       1       o
Y       *       :       1       1       u
@       *       :       1       1       sjwa
i       *       :       1       1       ie
y       *       :       1       1       uu
u       *       :       1       1       oe
a:      *       :       1       1       aa
e:      *       :       1       1       ee
e:      r       :       1       2       i
2:      *       :       1       1       eu
2:      r       :       1       2       u
o:      *       :       1       1       oo
o:      r       :       1       2       o
Ei      *       :       1       1       ei
9y      *       :       1       1       ui
Au      *       :       1       1       au
a:i     *       :       1       1       aai
o:i     *       :       1       1       ooi
ui      *       :       1       1       oei
iu      *       :       1       1       ieu
yu      *       :       1       1       uw
```

```
e:u     *       :       1       1       eeuw
E:      *       :       1       1       me
9:      *       :       1       1       meu
O:      *       :       1       1       mo
e~      *       :       1       1       e
a~      *       :       1       1       a
o~      *       :       1       1       o
9~      *       :       1       1       u
RULE_END
```

B.5 HMM File

```
~o
<STREAMINFO> 3 10 10 1
<VECSIZE> 21<NULLD><MFCC_E_D_N_Z>
~h "a"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<SWEIGHTS> 3
 1.000000e+000 1.000000e+000 1.000000e+000
<STREAM> 1
<MEAN> 10
 4.392169e+000 -1.835883e+001 -1.856646e+001 -4.802609e-001 9.296501e+000
 -8.347454e+000 3.038810e+000 6.059074e+000 -5.951744e+000 -1.426319e+001
<VARIANCE> 10
 8.435698e+001 1.562133e+002 2.103653e+002 2.028356e+002 4.088173e+002
 2.436477e+002 3.317146e+002 2.918913e+002 3.219072e+002 2.556351e+002
<GCONST> 7.283395e+001
<STREAM> 2
<MEAN> 10
 1.015623e+000 -5.022286e+000 -4.805974e+000 1.184052e+000 5.451186e+000
 2.730356e-001 5.598592e-001 1.025321e+000 -9.950465e-001 -3.224453e+000
<VARIANCE> 10
 9.612884e+000 1.314085e+001 1.562393e+001 2.125271e+001 2.571497e+001
 2.634945e+001 2.692418e+001 2.415101e+001 2.628845e+001 2.248129e+001
<GCONST> 4.840057e+001
<STREAM> 3
<MEAN> 1
 4.602402e-001
<VARIANCE> 1
 1.442221e-001
<GCONST> -9.852346e-002
<STATE> 3
<SWEIGHTS> 3
 1.000000e+000 1.000000e+000 1.000000e+000
<STREAM> 1
<MEAN> 10
 7.285262e+000 -2.291535e+001 -2.692610e+001 5.476334e+000 2.639052e+001
 -5.475887e+000 7.725285e-001 5.307328e+000 -1.154641e+000 -1.320635e+001
<VARIANCE> 10
 5.516068e+001 1.385840e+002 1.715508e+002 1.922638e+002 2.789575e+002
 2.356212e+002 2.233625e+002 2.867267e+002 2.563056e+002 2.588150e+002
<GCONST> 7.098734e+001
<STREAM> 2
<MEAN> 10
 5.743587e-001 9.277705e-001 8.465187e-002 3.744146e-001 1.041124e-001
 6.500685e-001 -5.524901e-001 -1.391898e+000 8.157665e-001 8.419775e-001
<VARIANCE> 10
 2.665063e+000 5.832265e+000 6.309563e+000 9.770234e+000 1.715404e+001
  1.201659e+001 1.405833e+001 1.410173e+001 1.518725e+001 1.278765e+001
<GCONST> 4.113078e+001
<STREAM> 3
<MEAN> 1
 -8.301759e-002
<VARIANCE> 1
 2.657933e-002
<GCONST> -1.789744e+000
<STATE> 4
<SWEIGHTS> 3
 1.000000e+000 1.000000e+000 1.000000e+000
<STREAM> 1
<MEAN> 10
 9.231950e+000 -1.050927e+001 -1.616354e+001 4.698371e+000 1.135131e+001
 -3.775125e+000 -3.478979e+000 -1.709562e+000 -2.035034e+000 -1.167247e+001
<VARIANCE> 10
 5.138276e+001 1.678958e+002 2.026178e+002 1.920019e+002 2.722980e+002
 2.211892e+002 2.342433e+002 2.830846e+002 2.740561e+002 2.852538e+002
<GCONST> 7.138498e+001
<STREAM> 2
<MEAN> 10
 -6.309538e-001 5.156171e+000 5.236431e+000 2.608254e-002 -3.991354e+000
 6.056778e-001 -9.733906e-001 -9.512351e-001 8.613650e-002 2.521105e+000
<VARIANCE> 10
```

```
 8.252870e+000 8.585522e+000 1.235152e+001 1.821269e+001 2.170505e+001
 1.966403e+001 2.213204e+001 1.997124e+001 2.123106e+001 1.867451e+001
<GCONST> 4.618559e+001
<STREAM> 3
<MEAN> 1
 -4.591587e-001
<VARIANCE> 1
 5.824743e-002
<GCONST> -1.005178e+000
<TRANSP> 5
 0.000000e+000 1.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 6.906633e-001 3.093368e-001 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 7.574322e-001 2.425678e-001 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 7.036940e-001 2.963060e-001
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
<ENDHMM>
```

B.6 VAMB 1.3 Configuration File

```
#==================================================================
# MODEL INFORMATION
#==================================================================
SAMPLE_FREQ                  = 11
LANGUAGE                     = DUTCH
MODEL_CATEGORY               = PHONEME
AMU_MODEL_CATEGORY           = PHONEMIC_BASED
MODEL_NAME                   = MONOPHONE
GENDER_TYPE                  = GENDER
HINIT_TRAIN_DATA             = AMSTERDAM_SABW
HEREST_TRAIN_DATA            = AMSTERDAM_SABW
HINIT_TRAIN_USE_FILE         = ---
HEREST_TRAIN_USE_FILE        = ---
HINIT_CORPUS                 = Default
HEREST_CORPUS                = Default
PROFILE_NAME                 = Female,Male
NOISE_KIND                   = Booth2
NOISE_SNR                    = 20
OPTION                       = Model_403
HINIT_LABEL                  = TIME_LABEL
HEREST_LABEL                 = TIME_LABEL
HREST_UPDATE_FLAG            = mvwt
HEREST_UPDATE_FLAG           = mvwt
HINIT_MAX_DATA               = 15000
HEREST_ITER                  = 10
BASE_OF_BIPHONE              = BEST,BEST
#==================================================================
# INFORMATION FILE PATH
#==================================================================
HMM_LIST_HINIT               = F:\VAMB1.3\Models\Model_403\HMMList.txt
HMM_LIST_HEREST              = F:\VAMB1.3\Models\Model_403\HMMList.txt
HMM_LIST_HEREST_NON_TRAINING = ---
CONFUSION_HMM_LIST           = ---
PROTO_INFO                   = F:\VAMB1.3\Models\Model_403\HMMProto.txt
UNI2ACE_RULE                 = ---
BIPHONE_INFO                 = ---
ELEMENT_COMP                 = F:\VAMB1.3\Models\Model_403\ElementComp.txt
MERGE_COMMAND                = ---
PHONEME_CONVERSION_RULE      = F:\VAMB1.3\Models\Model_403\P2SRules.txt
AMU_ELEMENT_COMP             = ---
GERBAGE_ELEMENTS             = ---
#==================================================================
# EVALUATION INFORMATION
#==================================================================
EVALUATION_MODE              = NORMAL
EV_DATA                      = AMSTERDAM_EVW
EV_NOISE_KIND                = Booth:20
EV_EDITCOM                   = Default
EV_LIST4NETWORK              = Default
EV_CORPUS                    = Default
EV_PERSON_SCRIPT_FILE        = ---
EV_VAD_LEVEL                 = NORMAL
EV_VORERO_CONFIG             = ---
#==================================================================
# GENERAL TRAINING DIRECTORY INFORMATION
#==================================================================
BASE_PATH                    = F:\VAMB1.3\Models\Model_403
PCM_PATH                     = F:\VAMB1.3\MAKE_MODEL\11kHz\DUTCH\PCM\TRAIN
EV_PCM_BASE                  = F:\VAMB1.3\MAKE_MODEL\11kHz\DUTCH\PCM\EV
LPC_PATH                     = F:\VAMB1.3\MAKE_MODEL\11kHz\DUTCH\LPC\TRAIN
EV_LPC_BASE                  = F:\VAMB1.3\MAKE_MODEL\11kHz\DUTCH\LPC\EV
NETWORK_PATH                 = F:\VAMB1.3\Models\Model_403\Network\TRAIN
EV_NETWORK_BASE              = F:\VAMB1.3\Models\Model_403\Network\EV
LABEL_PATH                   = F:\VAMB1.3\Models\Model_403
EV_LABEL_BASE                = F:\VAMB1.3\Models\Model_403\EV_LABEL
EV_CONFUSION_LABEL_BASE      = F:\VAMB1.3\Models\Model_403\EV_CONFUSION
MODEL_PATH                   = F:\VAMB1.3\Models\Model_403\MODEL
WORK_PATH                    = F:\VAMB1.3\Models\Model_403\WORK
ORG_CLEAN_PCM_LOCAL_PATH     = F:\VAMB1.3\MAKE_MODEL\OrgCleanPCM
ORG_NOISE_LOCAL_PATH         = F:\VAMB1.3\MAKE_MODEL\OrgNoise
ENG_DICT_PATH                = F:\VAMB1.3\Models\Model_403\Dictionary
WORD_LIST_PATH               = F:\VAMB1.3\CorpusList
```

```
ORG_MODEL_HMM_LIST           = F:\VAMB1.3\OrgModel\VAMB1.2_Model_400\HMMList.txt
ORG_MODEL_PATH               = F:\VAMB1.3\OrgModel\VAMB1.2_Model_400\Female,F:\VAMB1.3\OrgModel\VAMB1.2_Model_400\Male
ORG_MODEL_TYPE               = PHONEME              # PHONEME or WORD
ORGIN_OF_ALLLABEL            = ---
ORIGIN_OF_HEREST_MODEL_PATH  = ---,---
ORIGIN_OF_HMMPROTO_PATH      = ---,---
#================================================================
# TOOL PATH
#================================================================
HTK_PATH                     = F:\VAMB1.3\HTK_BIN
TOOL_PATH                    = F:\VAMB1.3\TOOLS
NOISEMIXER                   = F:\VAMB1.3\TOOLS\ConsoleTools\NoiseMixerC.exe
ACOUSTICANALYSIS             = F:\VAMB1.3\TOOLS\ConsoleTools\AcousticAnalysisC11kHz.exe
TRAINSCRIPTBUILDER           = F:\VAMB1.3\TOOLS\ConsoleTools\TrainingScriptBuilder.exe
HMMPROTOTYPE                 = F:\VAMB1.3\TOOLS\ConsoleTools\HMMPrototype.exe
CONVERT_UNI2ACE              = F:\VAMB1.3\TOOLS\ConsoleTools\HMMCHANGER.exe
EVALUATION_TOOL              = F:\VAMB1.3\TOOLS\ConsoleTools\getRecognition.pl
HINIT_DATA_CHECKER           = F:\VAMB1.3\TOOLS\ConsoleTools\HInitTrainScriptChecker.exe
LABEL_CHECKER                = F:\VAMB1.3\TOOLS\ConsoleTools\LabelChecker.exe
EV_EACH_WORD                 = F:\VAMB1.3\TOOLS\ConsoleTools\EachWordAccuracy.pl
EV_CALC_TIME                 = F:\VAMB1.3\TOOLS\ConsoleTools\EachWordCalcTime.pl
CONFUSION_MATRIX_TOOL        = F:\VAMB1.3\TOOLS\ConsoleTools\GetConfusionMatrix.pl
LABEL_MAKER                  = F:\VAMB1.3\TOOLS\ConsoleTools\LabelMaker\LabelMaker.pl
HTK_NETMAKER                 = F:\VAMB1.3\TOOLS\ConsoleTools\LabelMaker\HTKNetworkMaker.exe
LABEL_CHANGER_U2B            = F:\VAMB1.3\TOOLS\ConsoleTools\LabelMaker\UNI2ACE_LabelChanger.exe
LABEL_EDIT                   = F:\VAMB1.3\TOOLS\ConsoleTools\LabelMaker\LabelEdit.pl
NET_DRIVE                    = F:\VAMB1.3\TOOLS\ConsoleTools\NetDrive\NetDrive.exe
CHAR_CONVERT                 = F:\VAMB1.3\TOOLS\ConsoleTools\CharacterCodeConverter\csconv.exe
AMU_BUILDER                  = F:\VAMB1.3\TOOLS\ConsoleTools\AcousticModelUnitBuilderC.exe
VORERO_RECOGNITION           = F:\VAMB1.3\TOOLS\ConsoleTools\VoiceRecognitionCVer6_0_11kHz.exe
#================================================================
# NETWORK DATA INFORMATION (fixed information)
#================================================================
<CLEAN_DATA_INFO>
AMSTERDAM_EVW   \\Vorero-fs4\Clean\DUTCH\AMSTERDAM_48k\EVW      48000   0       0
AMSTERDAM_SABW  \\Vorero-fs4\Clean\DUTCH\AMSTERDAM_48k\SABW     48000   0       0
<NOISE_DATA_INFO>
Booth   \\Vorero-fs0\Clean\NoiseDataForVAMB\Booth_48k.pcm       48000
Booth2  \\Vorero-fs0\Clean\NoiseDataForVAMB\boothnoise11kHz.pcm 11025
```

Bibliography

[1] A. Acero, *Acoustical and Environmental Robustness in Automatic Speech Recognition*, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pennsylvania, September 1990

[2] A. Acero and R.M. Stern, *Environmental Robustness in Automatic Speech Recognition*, International Conference on Acoustics, Speech and Signal Processing, 1990, Albuquerque, NM, pp. 849-852

[3] D. Barker, *Microsoft Research Spawns a New Era in Speech Technology: Simpler, Faster, and Easier Speech Application Development*, PC AI Volume 16.6, 2003

[4] E. Baum and J.A. Eagon, *An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology*, Bulletin of American Mathematical Society, 1967, 73, pp. 160-363

[5] R.A. Cole, J. Mariani, H. Uszkoreit, A. Zaenen and V. Zue, *Survey of the State of the Art in Human Language Technology*, Center for Spoken Language Understanding CSLU, Carnegie Mellon University, Pittsburgh, PA., 1995

[6] S. Furui, *Recent Advances in Robust Speech Recognition*, ESCA-NATO Workshop on Robust Speech Recognition for Unknown Communication Channels, Pont-a-Mousson, France, pp. 11-20, April 1997

[7] S. Furui, *Steps Toward Flexible Speech Recognition*, Proc. 8th Australian International Conference on Speech Science and Technology, Canberra, Australia, pp. 19-29, 2000-12

[8] S. Furui, *Recent Advances in Spontaneous Speech Recognition and Understanding*, Proc. SSPR2003, Tokyo, Japan, pp. 1-6, 2003-4

[9] S. Furui, *Toward Robust Speech Recognition and Understanding*, TSD 2003, LNAI 2807, pp. 2-11, 2003

[10] S. Furui, *From Read Speech Recognition to Spontaneous Speech Recognition*, NLPRS 2001, pp. 19-25

[11] S. Furui, *Digital Speech Processing, Synthesis, and Recognition*, Second edition, revised and expanded, Marcel Dekker Inc., New York, 2001

[12] X. Huang, A. Acero and H. Hon, *Spoken Language Processing: a guide to theory, algorithm and system development*, 2001, Upper Sadle River, NJ, Prentice Hall

[13] K. Iwano, T. Seki, and S. Furui, *Noise Robust Speech Recognition Using Prosodic Information*, Proceedings Workshop on DSP in Mobile and Vehicular Systems, Nagoya, Japan, 2003-4

[14] L. Lamel, F. Lefevre, J. Gauvain, G. Adda, *Portability Issues for Speech Recognition Technologies*, Spoken Language Processing Group, CNRS-LIMSI, 91403 Orsay, France, 1999

[15] D. Liu, L. Nguyen, S. Matsoukas, J. Davenport, F. Kubala, R. Schwartz, *Improvements in spontaneous speech recognition*, Proceedings DARPA Broadcast News Transcription and Understanding Workshop, Lansdowne, VA, February, 1998, pp. 123-126.

[16] J. Makhoul and R. Schwartz, *State of the art in continuous speech recognition*, Proc. Natl. Acad. Sci. USA, Vol. 92, pp. 9956-9963, October 1995

[17] L. Rabiner, *The Power of Speech*, Science Magazine, Volume 301, September 12, 2003

[18] A.C.M. Rietveld and V.J. van Heuven, *Algemene Fonetiek*, Uitgeverij Coutinho, Bussum, 2de druk, 2001

[19] R. Rosenfeld, *Two Decades of Statistical Language Modeling: Where do we go from here?*, Proc. IEEE 88(8), 2000

[20] J. Sammon, *A Nonlinear Mapping for Data Structure Analysis*, IEEE Transactions on Computers, Vol. C-18. No. 5, May 1969

[21] M. Shozokai, *Acoustic Space Analysis Method Utilizing Multidimensional Scaling Technique*, Information Technology Laboratory, Asahi Kasei Corporation, Atsugi, Kanagawa, Japan, 2004

[22] M. Shozokai and G. Nagino, *Analysis of Speaking Styles by Two-Dimensional Visualization of Aggregate of Acoustic Models*, Information Technology Library, Asahi Kasei Corporation, Atsugi, Kanagawa, Japan, 2004

[23] M. Shozokai and G. Nagino, *Design of Ready-Made Acoustic Model Library by Two Dimensional Visualization of Acoustic Space*, Information Technology Library, Asahi Kasei Corporation, Atsugi, Kanagawa, Japan, 2004

[24] M. Stefik, *Strategic Computing at DARPA: Overview and Assessment*, Communications of the ACM, Volume 28, Number 7, July 1985, pp. 690-704

[25] C. Wang, *Prosodic Modeling for Improved Speech Recognition and Understanding*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001

[26] D. Willett, A. Worm, C. Neukirchen, G. Rigoll, *Confidence Measures For HMM-Based Speech Recognition*, 5th International Conference on Spoken Language Processing (ICSLP), Sydney, pp. 3241-3244, 1998

[27] S.J. Young, N.H. Russell, J.H.S. Thornton, *Token Passing: a Simple Conceptual Model for Connected Speech Recognition Systems*, Cambridge University Engineering Department, July 31, 1989

[28] S.J. Young, *Large Vocabulary Continuous Speech Recognition: a Review*, Cambridge University Engineering Department, April 8, 1996

[29] S.J. Young, *The HTK Book*, Cambridge University Engineering Department, December 2001

[30] S.R. Young, *Learning New Words from Spontaneous Speech: A Project Summary*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA., 1993