2006

Application of shifted delta cepstral features for GMM language identification

Jonathan Lareau

Application of Shifted Delta Cepstral Features for GMM Language Identification

Jonathan Lareau

October 10, 2006

Application of Shifted Delta Cepstral Features for GMM Language Identification

Jonathan Lareau

Thesis report submitted in partial fulfillment of the requirements of the degree:

Master of Science in Computer Science

Thesis Committee:

Adviser: Dr. Roger Gaborski _____

Reader: Dr. Carl Reynolds _____

Observer: Dr. Joe Geigel_____

Acknowledgments

I would especially like to thank my adviser Dr. Roger Gaborski for his commitment and patience while working with me on this thesis, as well as his willingness to encourage and provide ample opportunities for his students. My thanks to Dr. Reynolds for his attention to detail, and to Dr. Geigel for always allowing ample creative freedom and license. Special thanks also to my family and friends who have supported me and helped me keep my sanity throughout my educational career. Furthermore, I'd like to thank my fellow students in the labs here at RIT who have been instrumental in helping me brainstorm ideas and find solutions to some of the tough problems that have presented themselves in the course of my work.

Abstract

Spoken language identification (LID) in telephone speech signals is an important and difficult classification task. Language identification modules can be used as front end signal routers for multi-language speech recognition or transcription devices. Gaussian Mixture Models (GMM's) can be utilized to effectively model the distribution of feature vectors present in speech signals for classification. Common feature vectors used for speech processing include Linear Prediction (LP-CC), Mel-Frequency (MF-CC), and Perceptual Linear Prediction derived Cepstral coefficients (PLP-CC). This thesis compares and examines the recently proposed type of feature vector called the Shifted Delta Cepstral (SDC) coefficients. Utilization of the Shifted Delta Cepstral coefficients has been shown to improve language identification performance.

This thesis explores the use of different types of shifted delta cepstral feature vectors for spoken language identification of telephone speech using a simple Gaussian Mixture Models based classifier for a 3-language task. The OGI Multi-language Telephone Speech Corpus is used to evaluate the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The identification of the language spoken in a given speech signal is an important and active area of research in the speech processing community. Principle applications for such technology include correctly identifying the proper signal routing path for multilingual communication systems or as a preprocessor for multilingual speech recognition algorithms. It is common for language identification systems to need large sets of phonemically labeled data for training, usually one set for each language to be identified[48, 22, 28, 51, 50]. Such an approach to the Language Identification Problem is known as Phonemic Recognition followed by Language Modeling (PRLM)[28]. While PRLM algorithm are effective, the phonemic labeling required can be a laborious and time consuming process[51], and can also create extensibility issues due to the large amount of time and effort required to add new language models. It is therefore advantageous for systems to be developed that perform the discrimination task without this phonemic modeling prerequisite. Attempts at developing acoustic based models have resulted in the use of techniques such as Vector Quantization (VQ)[5, 6, 47, 32], Support Vector Machines(SVM)[46], and Gaussian Mixture Models (GMM)[20, 33, 35, 17].

This thesis will focus on the use of Gaussian Mixture Models due to their computational completeness, and their relative ease of use and readily available MATLAB implementations[18]. A simple classification architecture is used to evaluate the different types of feature vectors. For each language to be identified, a training set of speech utterances is used to build a GMM classifier. When testing an unknown utterance, a score corresponding to the probability of the utterance belonging to each Mixture Model is calculated, and the model with the highest score is chosen to be the language of the test utterance.

Any pattern recognition task will rely on a proper choice of features in order to perform well. Earlier experiments between Linear Predictive and Mel-Frequency Cepstral Coefficients have suggested that Linear Predictive coefficients may perform better than Mel-frequency based coefficients[3]. Recently, the proposal of using a shifting delta operation on the acoustic features of a speech signal for language identification with GMM's has produced promising results[2, 14], however the dependence on the type of features used to derive the shifted delta cepstra has not yet been discussed. This thesis explores this new technique in language identification tasks using the OGI Multi-language Telephone Data Corpus[49] and compares the different types of Shifted Delta Cepstral (SDC) features with previous features commonly used for language identification with Gaussian Mixture Models (GMM's).

This thesis makes use of the RASTA-MAT[4] software toolkit for some feature vector calculations as well as using the NETLAB[18] toolkit for basic GMM construction and manipulation.

1.1 Overview

The implementation and experiments discussed in this work form a modularized architecture for using Gaussian Mixture Models for classification purposes. A detailed discussion of the general architecture of the system is discussed in chapter 5, and source code and example script are presented in Appendix A and B. In this work, we specifically focus on classifying different languages from monaural audio recordings of telephone speech, and so we prelude the discussion on the system architecture by first introducing the relevant concepts and techniques that are utilized. Chapter 2 discusses the relevant physiological and mathematical background of speech and audio signals. Chapter 3 builds off of the material presented in chapter 2 in order to describe the different methods used for generating acoustic feature vectors and performing pre and post processing operations. In Chapter 4 we introduce the Gaussian Mixture Model that is used on the derived feature vectors for generating the experimental data presented in chapter 6.

Chapter 2

Background Material

This chapter aims to present relevant background material on the physiology of speech signals and basic mathematical techniques commonly used for speech data.

2.1 Physiology of Speech Signals

The human voice's role as a primary communication tool is undoubtedly one of the most important aspects of human intercommunication, consisting of a wide range of possible sounds that enables it to effectively transmit large amounts of information to its intended recipients. These produced sounds create a complex time-varying signal. Historically the primary recipient of speech signals has been other humans; however, with the advances made in computer technology speech signals can now be processed by electronic devices, provided that sufficient algorithms for parsing the signal content are developed. In working to develop such algorithms it is important to take into consideration the physical mechanisms that allow us as humans to communicate through speech.

Speech signals can be decomposed into different types of sounds; small units of a spoken language that are known as *phonemes*. These phonemes can be *voiced* or *unvoiced*. Voiced sounds are made while the *vocal folds* (also called the *vocal chords*) in the *larynx* (also known as the *voice-box*, or *Adams-apple*) are vibrating. The vocal folds are controlled by a set of muscles and cartilage which allows them to adapt their shape, and thereby change their vibration and the sounds that they produce. The vocal folds and the opening between them is called the *glottis*. The pathway in the head, by which the sound produced in the larynx travels, is referred to as the *vocal tract*.

All vowels and some consonants in the English language[1] are voiced sounds. Voiced consonants in English include: /b/ /d/ /g/ /v/ /TH/ /n/ /l/ /w/ /j/. Unvoiced sounds, like the name implies, include any sound made without the vibration of the vocal folds. Unvoiced consonants in the English language include: /p/ /t/ /k/ /s/ /h/.

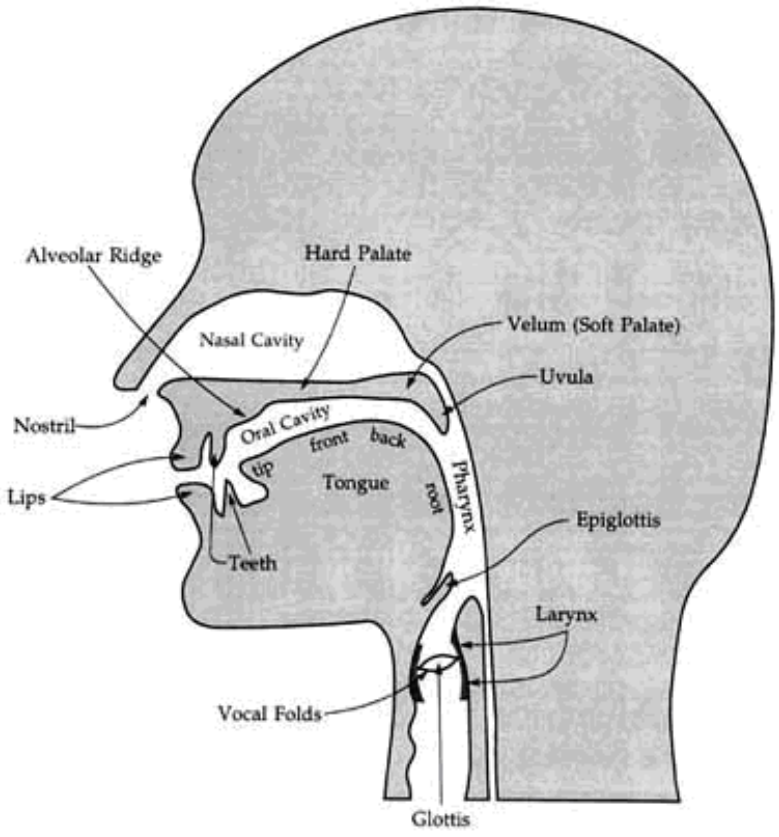

Figure 2.1: Physiology of speech production.
Source: www.jcarreras.homestead.com/RRPhonetics.html.

Obstruents are sounds which are made by obstructing the airway in some way which causes turbulence in the air flow. This turbulence in the air flow contributes to higher frequency noise in the speech signal due to the randomness imparted to the air stream from the turbulence. Although

[1] This document uses examples form the English language to illustrate academic concepts about the structure of speech and language. This is in light of the fact that this document is intended for publication in the United States, and so it is expected that the audience will be familiar with the sounds present in the English language.

obstruents are primarily unvoiced, voiced obstruents do occur. Obstruents can be subdivided into fricatives, plosives, and affricates. [43]

- Fricative sounds are generated by constricting the vocal tract at some point and pushing the air through the vocal tract at a high enough velocity to cause turbulence. An example of a fricative sound is the /f/ sound, where the lower lip is placed against the upper teeth in order to create turbulence in the air flow as air passes out the oral cavity.
- Plosive or stop sounds are created by blocking the vocal tract in some way, such as closing the lips and nasal cavity, in order to allow the buildup of air pressure behind the closure, and then suddenly releasing this pressure. Such a buildup and release can be observed in the /p/ (as in pat) sound.
- Affricates are sounds that begin like plosives but release as a fricative instead of releasing into a subsequent vowel sound. An example of such is the sound of the /ch/ in chicken.

Sonorants are speech sounds that are created without the use of turbulent airflow in the vocal tract. Vowels, approximates, nasal consonants, and flaps or taps are all sonorants. In opposition to obstruents, sonorants are primarily composed of voiced sounds.[45]

- Approximates are speech sounds that are in between a vowel and a consonant. In the English language, a common example is the 'y' in yes. The vocal tract is narrowed in the creation of an approximate, however not so much as to cause turbulence in the air flow.
- Nasal consonants are produced when the velum is lowered, which allows air to escape freely through the nose. The oral cavity still acts as a resonance chamber for the sound, however the air is passed out through the nasal cavity due to the blockage of the oral cavity by the tongue.
- Flaps or taps are consonantal sounds produced via a single contraction of the muscles so that one articulator is thrown against another. They are similar to plosive consonants except that flaps do not include a buildup of air, and therefore nor release or burst. The double 't' in the English word latter is a good example of a flap. The tongue strikes the roof of the mouth in such a way as to distinguish the sound but without a buildup of air.

Phonetic alphabets reference

The *IPA* column contains the symbol in the International Phonetic Alphabet, which is the alphabet used in many English dictionaries.

The *ASCII* column shows the corresponding symbol in the Antimoon ASCII Phonetic Alphabet, which can be used to type the pronunciation of words on a computer without the use of special fonts.

For a full description of the alphabets + audio recordings of the sounds, visit **www.antimoon.com/ipa**

vowels

IPA	ASCII	examples
ʌ	^	cup, luck
ɑ:	a:	arm, father
æ	@	cat, black
ə	..	away, cinema
e	e	met, bed
ɜ:	e:	turn, learn
ɪ	i	hit, sitting
i:	i:	see, heat
ɒ	o	hot, rock
ɔ:	o:	call, four
ʊ	u	put, could
u:	u:	blue, food
aɪ	ai	five, eye
aʊ	au	now, out
oʊ əʊ	Ou	go, home
eə	e..	where, air
eɪ	ei	say, eight
ɪə	i..	near, here
ɔɪ	oi	boy, join
ʊə	u..	pure, tourist

consonants

IPA	ASCII	examples
b	b	bad, lab
d	d	did, lady
f	f	find, if
g	g	give, flag
h	h	how, hello
j	j	yes, yellow
k	k	cat, back
l	l	leg, little
m	m	man, lemon
n	n	no, ten
ŋ	N	sing, finger
p	p	pet, map
r	r	red, try
s	s	sun, miss
ʃ	S	she, crash
t	t	tea, getting
tʃ	tS	check, church
θ	th	think, both
ð	TH	this, mother
v	v	voice, five
w	w	wet, window
z	z	zoo, lazy
ʒ	Z	pleasure, vision
dʒ	dZ	just, large

special symbols

IPA	ASCII	meaning
ˈ	'	' is placed before the stressed syllable in a word. For example, the noun *contract* is pronounced /'kɒntrækt/, and the verb *to contract* is pronounced /kən'trækt/.
ʳ	(r)	/ka:ʳ/ means /ka:r/ in American English and /ka:/ in British English.
i	i(:)	/i/ means /i:/ or /ɪ/ or something in between. Examples: *very* /'veri/, *ability* /ə'bɪlɪti/, *previous* /'pri:viəs/.
ᵊl	.l	/ᵊl/ shows that the consonant /l/ is pronounced as a syllable. This means that there is a short vowel (shorter than the /ə/ sound) before the consonant. Examples: *little* /'lɪtᵊl/, *uncle* /'ʌŋkᵊl/.
ᵊn	.n	/ᵊn/ shows that the consonant /n/ is pronounced as a syllable. Examples: *written* /'rɪtᵊn/, *listen* /'lɪsᵊn/.

Figure 2.2: Phonetic Alphabet Chart.
Source: www.Antimoon.com

2.1.1 The Source-Filter Conceptual Model

From a signal processing standpoint, a speech signal can be thought of as containing two main components; the *formants* and the *excitation signal*. A *formant* is a peak in the frequency spectrum of a speech signal which results from the resonant frequencies determined by the vocal tract shape when producing a specific sound[7]. Often the vocal tract is conceptually thought of as a hollow non-uniform acoustic tube with a time varying area function, at one end is the larynx which produces the sound, and the other end is representative of the opening at the mouth. An *excitation signal* is generated via air flowing from the lungs and passing through the larynx. This excitation signal acts as a generating source which travels through the vocal tract. As the excitation signal passes through various areas of the acoustic tube, it is filtered due to the different resonances caused by the pathway's area and shape.

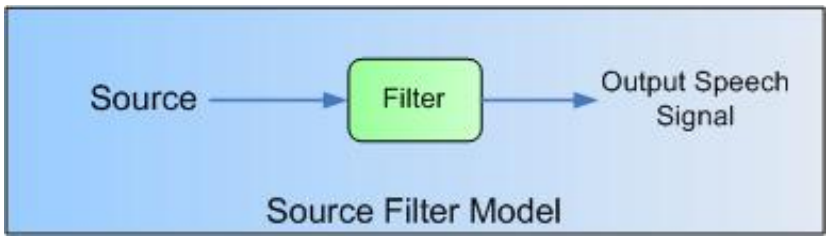

Figure 2.3: Conceptual Block Diagram of the Source-Filter Model.
The 'source' is the excitation signal produced by the airflow through the voice-box, and the 'filter' is derived from the resonant frequencies of the vocal tract.

2.2 Mathematical Techniques and Tools for Speech Signals

This section briefly defines and discusses common mathematical techniques for speech signal processing. It is assumed that the reader has previous knowledge of calculus, differential equations, Laplace, and z-transforms, but that the reader may not be familiar with advanced signal processing and engineering techniques such as convolution, auto-correlation, Fourier transforms & analysis, and homomorphic signal processing.

2.2.1 Convolution

Convolution is a mathematical operation that expresses the amount of overlap between two signals as one of the signals is passed over the other[37, 31]. The convolution operation is denoted by the $\otimes$ symbol.

$$x(t) \otimes h(t) = \int x(\tau)h(t-\tau)d\tau$$

The source-filter model of speech production can be mathematically thought of as the convolution of the excitation signal $x(t)$ with the formant filter impulse response signal $h(t)$.

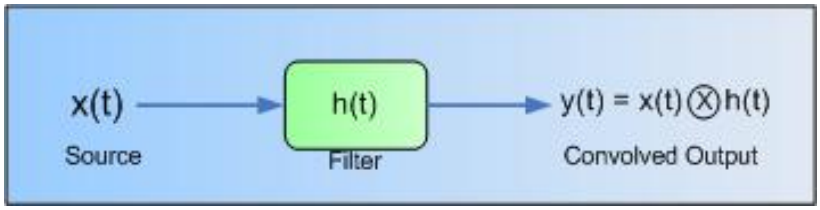

Figure 2.4: The Source-Filter model of speech production as a Convolution Operation

2.2.2 Auto-correlation

The *auto-correlation* function of a continuous real signal $x(t)$ is:

$$R_x(t) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} x(\tau)x(t+\tau)d\tau$$

For a complex function $x(t)$, the auto-correlation function is defined as

$$\rho_x(t) = \bar{x}(-t) \otimes x(t)$$

$$= \int_{-\infty}^{\infty} x(t+\tau)\overline{x}(\tau)d\tau$$

where $\overline{x}$ denotes the complex conjugate. For a complex number $x = a + bi$, the complex conjugate is given as $\overline{x} = a - bi$. Here $i = \sqrt{-1}$.[40, 31]

The auto-correlation function is maximum at the origin, whose value is equivalent to the power of the signal[31]. In many cases, this initial maximum is ignored or marginalized while subsequent maximums are deemed of interest. For example, a simplistic auto-correlation based pitch extraction technique could use this term for normalization purposes, and then search for the next peak that is over a specified threshold.[29] In this way the peak corresponding to the most prominent pitch periodicity can be found and the pitch period estimated.

2.2.3 Fourier Analysis

Fourier decomposition can represent a data sequence as a linear combination of a set of sine and cosine basis functions. From these basis functions the signal can be completely reconstructed provided that the sampling rate is high enough to avoid aliasing effects. The time domain signal is decomposed into a set of amplitudes and associated periodicities. The independent variable associated with the Fourier spectrum of a signal is called *frequency*, and it has a unit of *Hertz*, which is equivalent to $(\frac{cycles}{second})$. The product of frequency and time units is dimensionless, meaning that they are reciprocally related. For example, a sine wave whose period is $T = 50\,ms = \frac{50\,ms}{1\,cycle} = \frac{.050\,seconds}{1\,cycle}$ has a frequency of $f = \frac{1\,cycle}{.050\,seconds} = 20\,hertz$. The Fourier transform is used to convert a signal representation from the time domain into the frequency domain and vice-versa.

The Fourier transform pair for a time-domain signal $f(x)$ is given by[39, 31]:

$$f(x) = \mathcal{F}^{-1}[F(k)] = \int_{-\infty}^{\infty} F(k)\, e^{2\pi i k x} dk$$

$$F(k) = \mathcal{F}\,[f(x)] = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x} dx$$

or, in terms of angular instead of oscillation frequency:

$$f(t) = \mathcal{F}^{-1}[F(\omega)] = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)\, e^{i\omega t} d\omega$$

$$F(\omega) = \mathcal{F}\,[f(t)] = \int_{-\infty}^{\infty} f(t)\, e^{-i\omega t} dt$$

where $\omega = 2\pi f$ and is in terms of $(\frac{radians}{second})$, and f is the frequency of oscillation in Hertz. When plotted as a graph, the Fourier spectrum $F(\omega)$ of a signal is usually viewed on the Decibel scale,

which is given by:

$$F_{dB}(\omega) = 20 * log_{10}\left[\,|\,F(\omega)\,|\,\right]$$

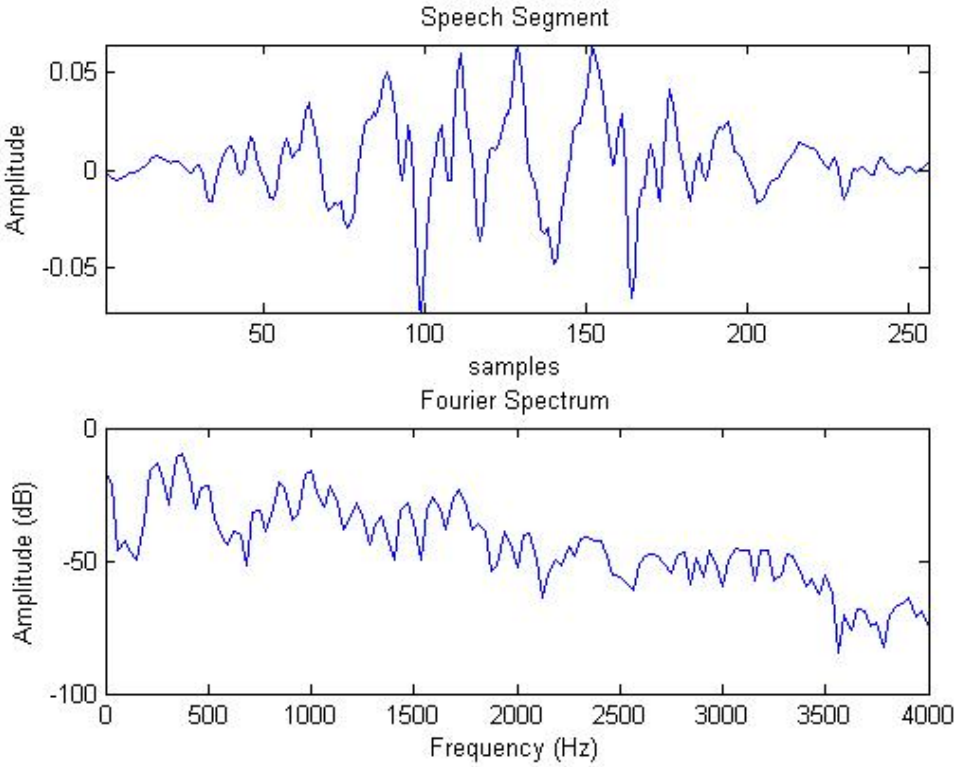

Figure 2.5: Speech Signal and its Associated Fourier Spectrum

The Fourier transform breaks down a signal into a set of additive sine and cosine components. By using sinusoids as the basis function, a compact and information rich representation of an input signal can be achieved. It should be noted that the use of Fourier Analysis is not restricted to time-frequency variable pairs, but is valid for any set of variables (x, y) whose product is dimensionless.[31]

When objects such as guitar strings or the vocal chords are vibrating, the signal produced contains many different natural vibration frequencies. This is due in part to the fact that the endpoints are essentially held stationary, and standing waves must be generated in the object in order for vibration to occur. These different natural frequencies are known as *fundamental* and *harmonic* frequencies.

As a simple visual description:

> First, consider a guitar string vibrating at its natural frequency or harmonic frequency. Because the ends of the string are attached and fixed in place to the guitar's structure (the bridge at one end and the frets at the other), the ends of the string are unable to move. Subsequently, these ends become nodes - points of no displacement. In between these two nodes at the end of the string, there must be at least one anti-node. The most fundamental harmonic for a guitar string is the harmonic associated with a standing wave having only one anti-node positioned between the two nodes on the end of the string. This would be the harmonic with the longest wavelength and the lowest frequency.[9]

The fundamental frequency is the lowest vibration frequency component, and the harmonic frequency components of the sound wave occur near integer multiples of the fundamental[2]. Harmonic frequencies of a signal can be seen as regularly spaced peaks in the signal's Fourier spectra. The term 'Even Harmonics' refers to the contributions of cosine waves, because the cosine waveform is a mathematically even function. Similarly, the term 'Odd Harmonics' refers to the harmonics due to sinusoidal components, because the sinusoidal waveform is a mathematically odd function.

[2]We can think of the fundamental frequency as being the initial harmonic frequency. Henceforth, we shall simply refer to the harmonics of a speech signal.

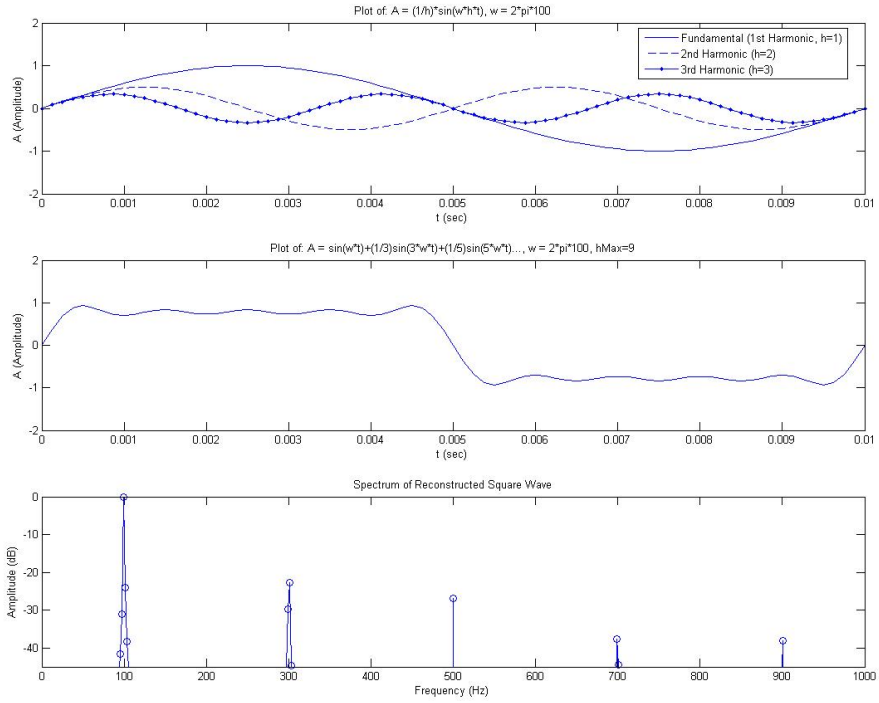

Figure 2.6: Graphical depiction of fundamental and harmonic sinusoids

- (Top) Plot of the first three harmonics with fundamental frequency of 100 Hz. $A = \frac{1}{h}sin(h\omega t)$, $\omega = 2\pi 100$, $h = [1, 2, 3]$
- (Middle) Approximation of a square wave signal obtained by adding sequential odd harmonics. $A = \sum_{h=1,3,...}^{h=9} \frac{1}{h}sin(h\omega t)$, $\omega = 2\pi 100$
- (Bottom) The Fourier spectrum of the square wave approximation.

2.2.3.1 The Discrete Fourier Transform

The *Discrete Fourier Transform (DFT)* is the digital equivalent of the continuous Fourier Transform equations presented above. A sequence of N complex numbers $x_0, ..., x_{N-1}$representing a discrete input signal is turned into the sequence of N complex numbers $X_0, ..., X_{n-1}$representing that signal's Fourier coefficients according to the formula[42, 31]:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, ..., N-1$$

The inverse formula is:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \quad n = 0, ..., N-1$$

2.2.3.2 The Fast Fourier Transform

The *Fast Fourier Transform (FFT)* is an efficient algorithm for calculating the Discrete Fourier Transform (DCT)[31, 38]. It is a common tool used for analyzing quantized signals, and is built into many mathematical tool sets such as MATLAB. While the DFT requires approximately N^2complex multiply and add operations, the FFT only executes in the order of $N \log_2 N$ similar operations, where N is the number of samples. [31]

2.2.3.3 The Discrete Cosine Transform

A related technique, the *Discrete Cosine Transform (DCT)*, is equivalent to the DFT of roughly twice the length, operating on real data with even symmetry[41]. Conceptually, it can be thought of as only computing the even half of the full Fourier Transform of an input signal. It is primarily used for compression techniques, such as the JPEG compression algorithm[24], due to the empirical observation that it is better at concentrating energy into lower order coefficients than the DFT. The one dimensional DCT is given by the formula[12, 36]:

$$X_k = w_k \sum_{n=1}^{N} x_{n-1} cos\left[\frac{\pi}{N}\left(n - \frac{1}{2}\right)k\right] \quad k = 0, ..., N-1$$

$$w_k = \left\{ \begin{array}{ll} \frac{1}{\sqrt{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq k < N \end{array} \right\}$$

with inverse:

$$x_{n-1} = \sum_{k=0}^{N-1} w_k X_k cos\left[\frac{\pi}{N}\left(n - \frac{1}{2}\right)k\right] \quad n = 1, ..., N$$

$$w_k = \left\{ \begin{array}{ll} \frac{1}{\sqrt{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq k < N \end{array} \right\}$$

2.2.3.4 An Illustrative MATLAB Example of Fourier Analysis

The following MATLAB 7.0 help file example code illustrates the use of the Fourier transform using the Fast Fourier Transform (FFT) to find the frequency components of a noise corrupted signal:

```
t = 0:0.001:0.6;
```

```
x = sin(2*pi*50*t)+sin(2*pi*120*t); %create the clean signal
y = x + 2*randn(size(t)); %corrupt with noise
figure, subplot(2,1,1), plot(1000*t(1:50),y(1:50));
title('Signal Corrupted with Zero-Mean Random Noise');
xlabel('time milliseconds)') ;

Y = fft(y,512);
Pyy = Y.* conj(Y) / 512;
f = 1000*(0:256)/512;
subplot(2, 1, 2), plot(f,Pyy(1:257));
title('Frequency content of y');
xlabel('frequency (Hz)');
```

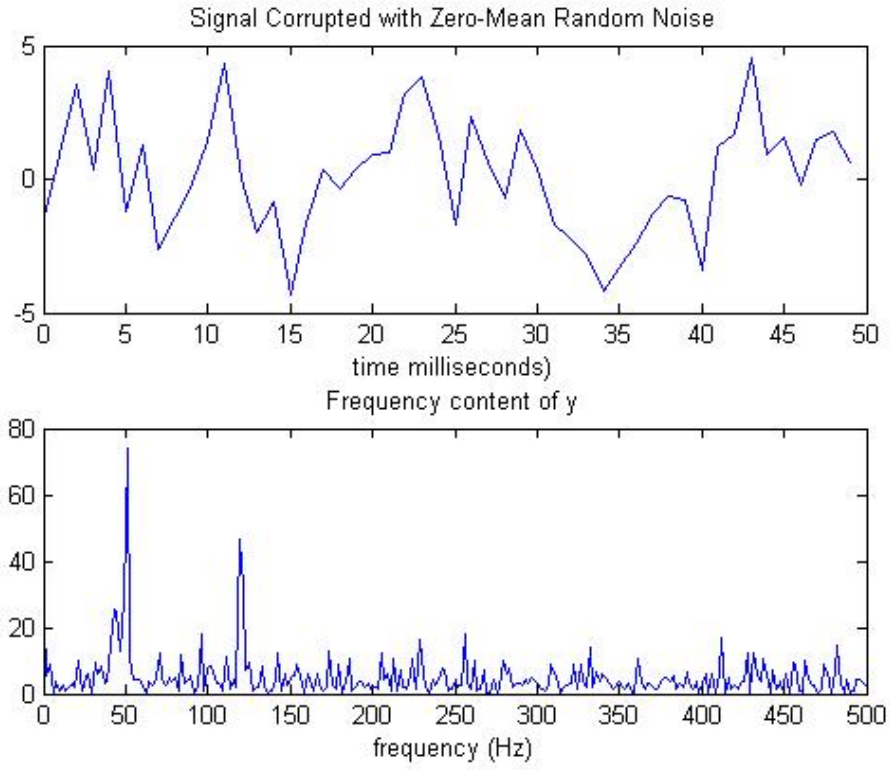

Figure 2.7: Output from MATLAB Fourier example code.

The first section of the example code creates a test signal consisting of the summation of two sinusoids at different frequencies, and then corrupts the signal with additive random noise. The second portion of the example code uses the Fast Fourier Transform to find the spectrum of the signal. The two large spikes that are visible in the spectrum plot represent the contributions of the two component sinusoids.

2.2.3.5 The Convolution Property of the Fourier Transform and its application to speech signals

One property of the Fourier transform is that *a convolution operation in either the time or frequency domain reduces to multiplication in the other domain.* This relationship greatly simplifies numerical manipulation of the source-filter speech model. The proof is as follows[31]:

$$\mathcal{F}[f(t) \otimes g(t)] = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \right] e^{-i\omega t} dt$$

Changing the order of integration:

$$= \int_{-\infty}^{\infty} f(\tau) \left[\int_{-\infty}^{\infty} g(t-\tau)\, e^{-i\omega t} dt \right] d\tau$$

The time shifting property (see proof below) of the Fourier transform states that $\mathcal{F}[g(t-t_0)] = G(\omega)e^{-j\omega t_0}$, hence we can write:

$$= \int_{-\infty}^{\infty} f(\tau) \left[G(\omega) e^{-i\omega\tau} \right] d\tau$$

$$= G(\omega) \int_{-\infty}^{\infty} f(\tau)\, e^{-i\omega\tau}\, d\tau$$

$$= G(\omega)\, F(\omega) = \mathcal{F}[g(t)]\, \mathcal{F}[f(t)]$$

Proof of the time shifting property of the Fourier Transform:

$$\mathcal{F}[g(t-t_0)] = \int_{-\infty}^{\infty} g(t-t_0)\, e^{-i\omega t} dt$$

We change the variable of integration, let $x = (t - t_0)$

$$= \int_{-\infty}^{\infty} g(x)\, e^{-i\omega(x+t_0)} dx$$

$$= \int_{-\infty}^{\infty} g(x)\, e^{-i\omega t_0}\, e^{-i\omega x} dx$$

$$= \left[\int_{-\infty}^{\infty} g(x)\, e^{-i\omega x} dx \right] e^{-i\omega t_0}$$

$$= G(\omega)\, e^{-j\omega t_0}$$

To use the convolution property of the Fourier transform with the source-filter model of speech, we can proceed as follows; A string of pulses located at the harmonic frequencies can represent the excitation component of the speech signal $F(\omega)$. This signal can then be multiplied by an envelope $H(\omega)$ representing the formant filter. The time-domain speech signal $y(t)$ is then the inverse Fourier transform of $Y(\omega) = F(\omega)H(\omega)$.

2.2.3.6 Using the Fourier Transform to find the Auto-correlation function

The Fourier transform can also be used to calculate the auto-correlation of an input signal by using the *Wiener-Khinchin* Theorem[38, 31], which states that the auto-correlation is equivalent to the Fourier transform of the absolute square of the Fourier spectrum of a signal $x(t)$.

$$\rho_x(t) = \mathcal{F}[\,|\,\mathcal{F}[x(t)]\,|^2]$$

2.2.3.7 The Short Time Fourier Transform, Spectrograms, and Speech

Applying a Fourier transform to a time-domain signal inherently incurs the loss of temporal information about that signal. When dealing with long duration, quasi-periodic signals such as speech, it is often desired to retain some semblance of the temporal information while examining the frequency components of a signal. A *spectrograph*, also called a *spectrogram*, allows for easy visualization of both the temporal and frequency structure of speech or other signals. A spectrogram is an image representation of the *Short Time Fourier Transform* (STFT)[44, 26] of a signal. The Short Time Fourier Transform of a signal provides a means of joint time-frequency analysis. The input signal

is broken up into successive time frames and the Fast Fourier Transform (FFT) of the input signal at each frame is computed.

2.2.4 Use of Hamming Window

It is common practice, but not completely necessary, to overlap and weight each of these signal frames so that the endpoints of each frame are near zero, and so that when summed back together the overlapped frames add back to the original signal. The common method is to overlap each frame by 1/2 of the frame size, and to apply a *Hamming*[19] window to the frame samples. The use of the Hamming window can ensure smooth frame to frame transitions when used in overlapping analysis[22]. The Hamming window can be defined as:

$$w[k+1] = 0.54 - .46cos\left[2\pi\frac{k}{N-1}\right] \quad k = 0, ..., N-1$$

where N is the number of samples in each frame.

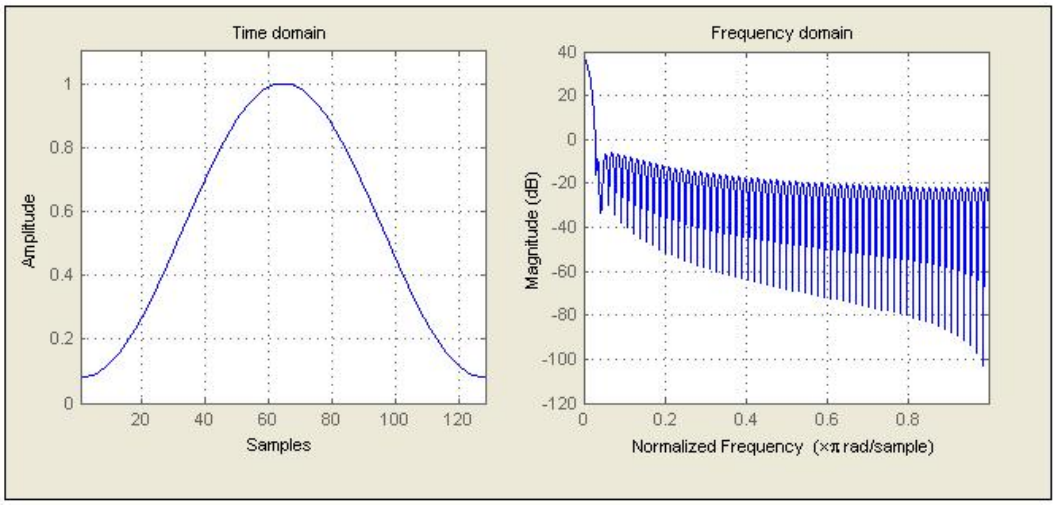

Figure 2.8: The Hamming Window function for N = 128.
Graphic obtained by using the MATLAB command: >> wvtool(hamming(128))

The results of this operation (with, or without windowing) are then viewed, usually as a color coded plot according to amplitude with time slices on the independent axis and frequency bins on the dependent axis.

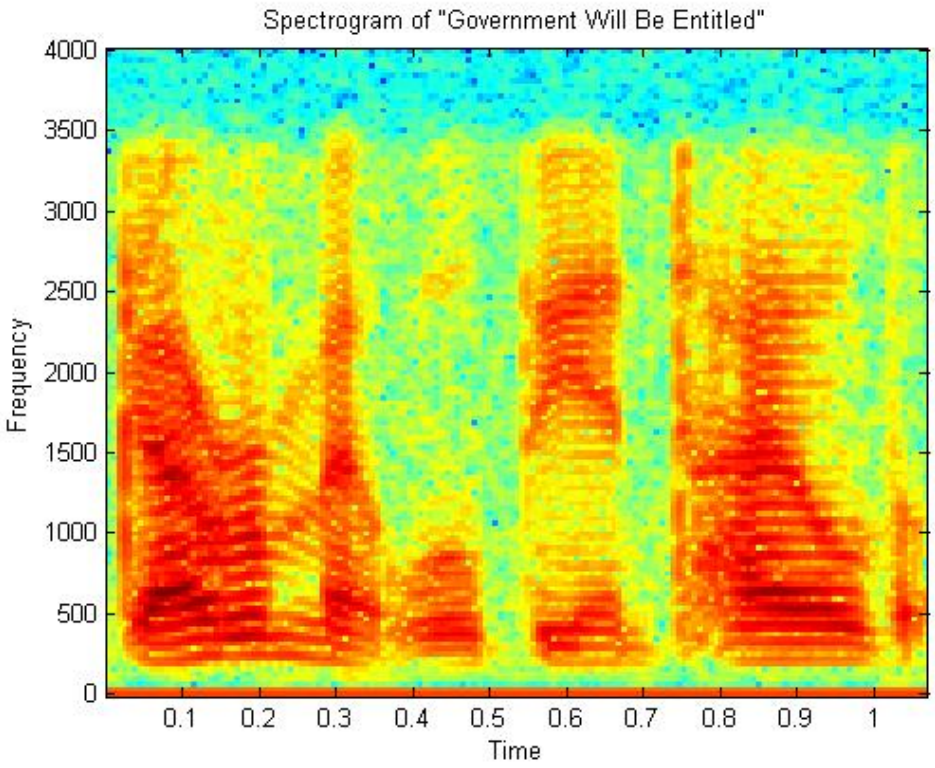

Figure 2.9: Spectrogram of an example telephone speech signal.
The predominantly horizontal bands correspond to the contributions from the harmonics of the excitation signal. The larger, more slowly varying amplitude envelope in each frame is due to the formants. High amplitude values are shown as red, and low amplitude values are shown as blue.

2.2.5 Homomorphic Signal Processing and the Cepstrum

An observation on the Fourier transform of speech signals, is that the log-spectrum itself is highly periodic during voiced frames of the speech signal. The amplitude envelope of the harmonics component in the voiced frames oscillates much more rapidly than the envelope due to the formants. The *Cepstrum* of a signal is historically derived from Fourier spectrum[3], and reveals the contributions of these two different components of the speech signal. This type of processing of speech signals, which is based on the principle of superposition of the formant and harmonic components, is a form of Homomorphic signal processing.[22]

[3]Hence its name, simply reverse the first four letters of *spec*trum.

It should be noted here that there are various ways to obtain the cepstrum of a signal, depending on the parameterization desired. In some cases spectral warping is applied to the FFT derived spectrum first in order to remove channel effects, or to emulate psycho-acoustic phenomenon. In other cases an approximation procedure is used to first find the formant envelope before deriving the Cepstral coefficients. These different types of spectral approximations result in different Cepstra, and are individually discussed later in this thesis (see sec. 3). In all cases the Cepstrum corresponds to an encoding of the signal that allows the formant and harmonic contributions to be easily separable. For the current conceptual discussion and definition of the Cepstrum, we use the Fourier spectrum derived Cepstral Coefficients.

In computing the cepstrum, the spectrum of the speech signal is treated as an input signal in and of itself. Computation of the cepstrum consists of taking the inverse Fourier transform of the logarithm of the absolute spectrum of the speech signal.[26]

$$X(q) = \mathcal{F}^{-1}[log(\,|\mathcal{F}[x(t)]\,|\,)]$$

This has the effect of decomposing the logarithm of the absolute value of the spectrum of the speech signal into a set of sine and cosine basis functions.[4]

The independent variable of a cepstral graph is a measure of time units called *quefrency*, whose name comes from the manipulation of the spectral unit of frequency.

[4] It should be noted that it is often the case that since the cepstral coefficients are to be used as feature vectors, that the discrete cosine transform is utilized instead of the Fast Fourier Transform due to the DCT's inherent compression characteristics. In this section, which is intended to introduce the ideas and concepts of the Cepstrum, we use the conceptually simpler and more historically accurate method of using the full Fourier transform for Cepstrum computation.

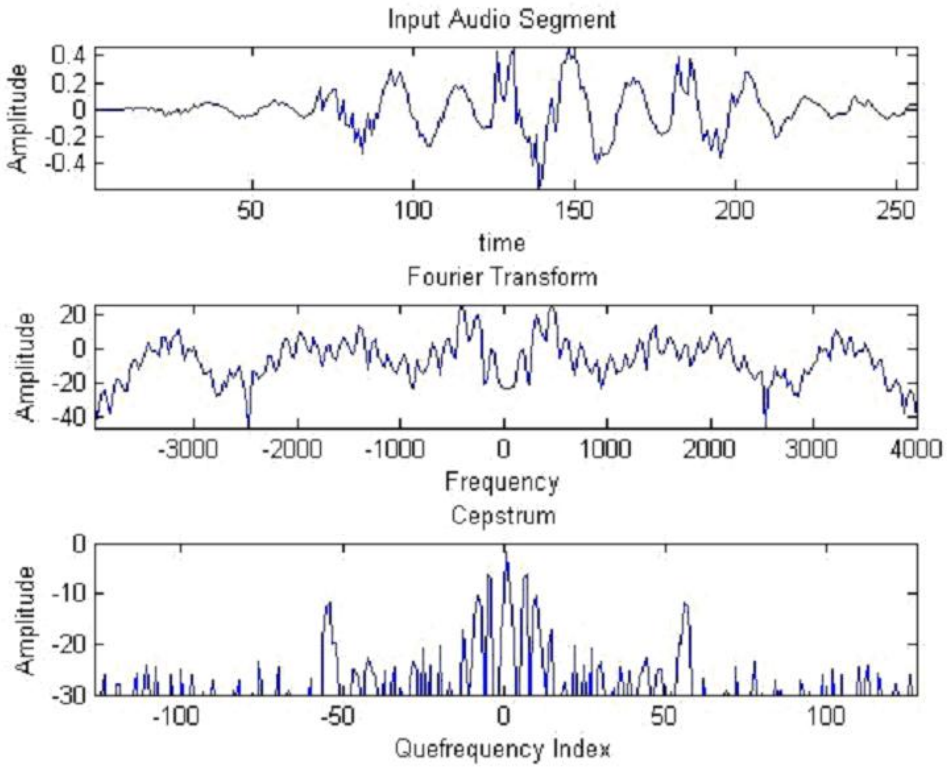

Figure 2.10: The Cepstrum of an Example Speech Signal.
(Top) Input Audio Segment with Hamming window applied. (Middle) Fourier Spectrum of the audio. (Bottom) The Cepstrum of the audio signal segment. The large peaks near the center (near index 0) are due to the formant envelope of the speech signal, whereas the two large peaks located near ± 50 are due to the periodicities present in the excitation component of the speech signal.

By retaining only a small number (12 is a common value) of the beginning Cepstral Coefficients of a signal as a feature vector, only the low quefrency components corresponding to the formant components of the signal are used. For speech and language recognition tasks this is desirable because the formants of a speech signal convey a large portion of the characteristic information of the phonemes produced. Thus, using the beginning Cepstral coefficients of a signal can provide a compact and useful encoding for signal processing tasks.[5]

[5] It should also be noted here that since the first Cepstral coefficient amplitudes corresponding to the formants drop off rather rapidly, it is common practice to arbitrarily weight these coefficients when they are used in speech recognition tasks in order to avoid round off errors and the like. Also, since the initial Cepstral Coefficient is indicative of the power of the signal, which is a varying parameter not necessarily related to the language being spoken, this thesis does not utilize the first cepstral coefficient in its experiments.

Chapter 3

Calculation of Different Feature Vectors for Speech Signals

In this section we discuss the calculation of the different types of feature vectors compared in this thesis; Mel Frequency derived Cepstral Coefficients (MF-CC's), Linear Predictive Cepstral Coefficients (LP-CC's), Perceptual Linear Predictive Cepstral Coefficients (PLP-CC's), along with Shifted Delta versions of the same (SD-MF-CC's, SD-LP-CC's, and SD-PLP-CC's, respectively). The discussion first focuses on the universal pre-processing steps taken for all data. Feature vector calculation using psycho-acoustic scaling and linear prediction is then discussed, as well post-processing with Cepstral Mean Subtraction and the Shifting Delta operation.

The MATLAB routines used in this thesis for calculating the different cepstral feature vectors are based off of the RASTA-MAT toolbox[4] written by Dan Ellis. The RASTA-MAT toolbox contains routines for computing LP,MF,PLP and RASTA feature vectors, as well as routines for converting between coefficient types, calculating perceptual filter banks, and computing delta coefficients, among others.

3.1 Emphasis Filters for Pre-Processing

In an effort to keep the signal to noise ratio of the speech signal's high, it is common practice to use a pre-emphasis Finite Impulse Response filter (FIR) in speech processing algorithms.

$$H_{pre}(z) = 1 + a_{pre}z^{-1}$$

Where the range of a_{pre} is typically $[-1.0, -0.4]$. The spectrum of voiced speech has a natural occurring attenuation of approximately 20dB/decade, or equivalently 6dB/octave, due to the physiology of speech production. The pre-emphasis filter serves to flatten the spectrum of the speech signal, in turn emphasizing the higher formant components.[22]

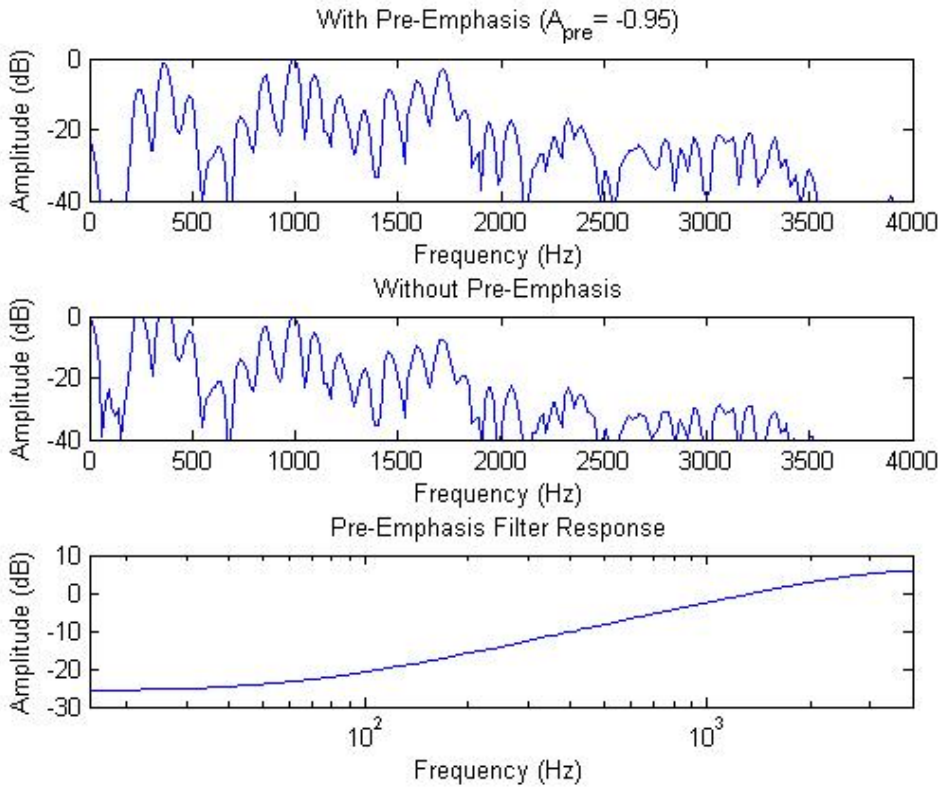

Figure 3.1: Pre-Emphasis filtering.
(Top) Spectrally flattened speech spectrum using $a_{pre} = -.95$. (Middle) Original speech spectrum. (Bottom) Pre-Emphasis Filter Response. The filter serves to compensate for the 20dB/decade attenuation that naturally occurs during speech production.

3.2 Cepstral Enhancement of OGI Telephone Speech Database

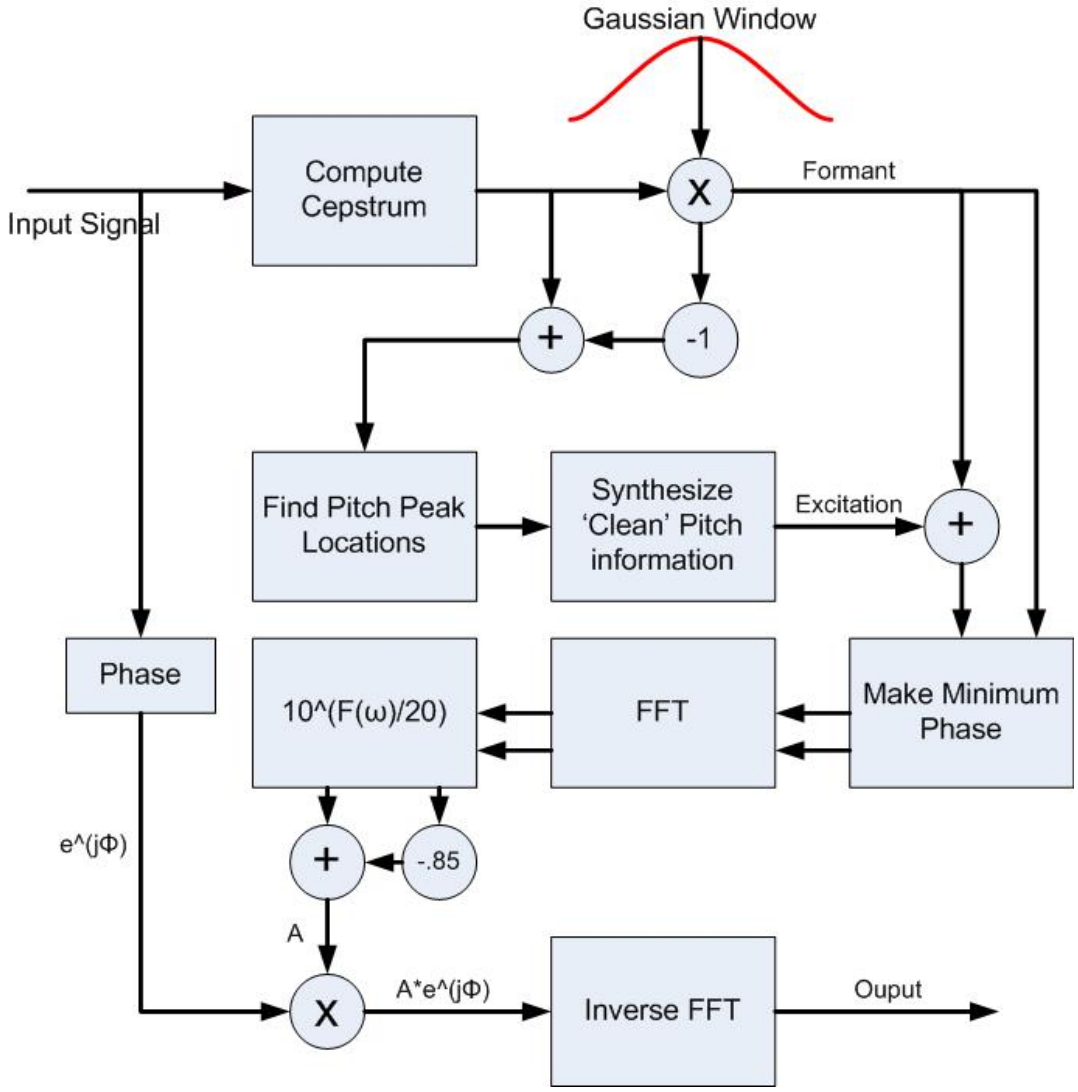

Figure 3.2: Cepstral Speech Enhancement Algorithm Flowchart

For this thesis a heuristic Cepstrum based Speech Enhancement algorithm was utilized in order to improve the sound quality of the telephone speech signals. The intent was to boost the $\frac{Speech\ Signal}{Channel\ Background\ Noise}$ ratio.

The formant component is first isolated by using a Gaussian window centered on the low quefrency components of the Cepstrum of the signal. This formant component is then subtracted from the full Cepstrum, and the locations of excitation signal peaks are then identified via thresholding and peak-picking, and a new cepstrum is created. Inverting the cepstrum then results in an estimate

of the spectrum envelope. Finally, an arbitrary mixing factor ($-.85$) is applied between the re-generated spectrum and the spectral formant envelope, the original signal phase is added back, and an inverse Fourier transform produces the output signal. In qualitative listening tests the enhancement algorithm performed well, and the incorporation of the speech enhancement algorithm in combination with Pre-Emphasis filtering and Cepstral Mean Subtraction (see sec. 3.5) in our Language Identification trials has been shown to generally improve performance.

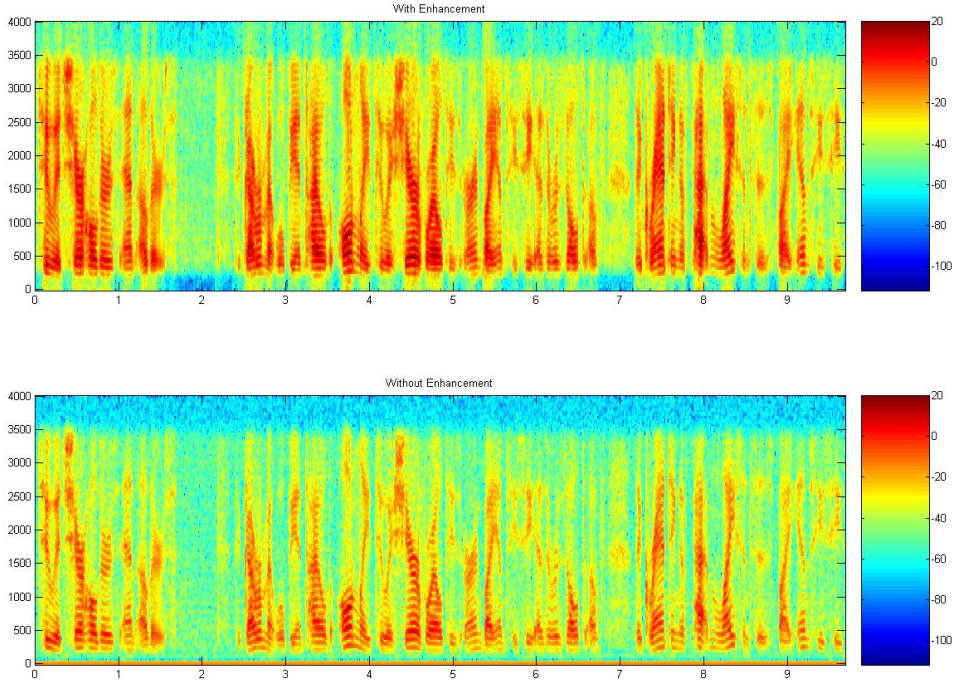

Figure 3.3: Cepstral Speech Enhancement sample results.
(Top) Speech Enhancement Algorithm output. (Bottom) Original Telephone speech signal. As can be seen by comparing the two Spectrograms, the enhanced speech signal is more pronounced.

3.3 Psycho-acoustics and The Mel Frequency Scale

Perceptual scaling of the frequency components of an audio signal is commonly used to approximate the response of the human ear to acoustic input. It has been experimentally confirmed that humans perceive a warped version of the true frequency (pitch) of audio signals[30, 25]. The scale derived from these observations that relates perceived frequency to the actual physical frequency[34], is

called the *Mel Scale:*

$$mel\ frequency = 2595\, log_{10}(1 + \frac{f}{700})$$

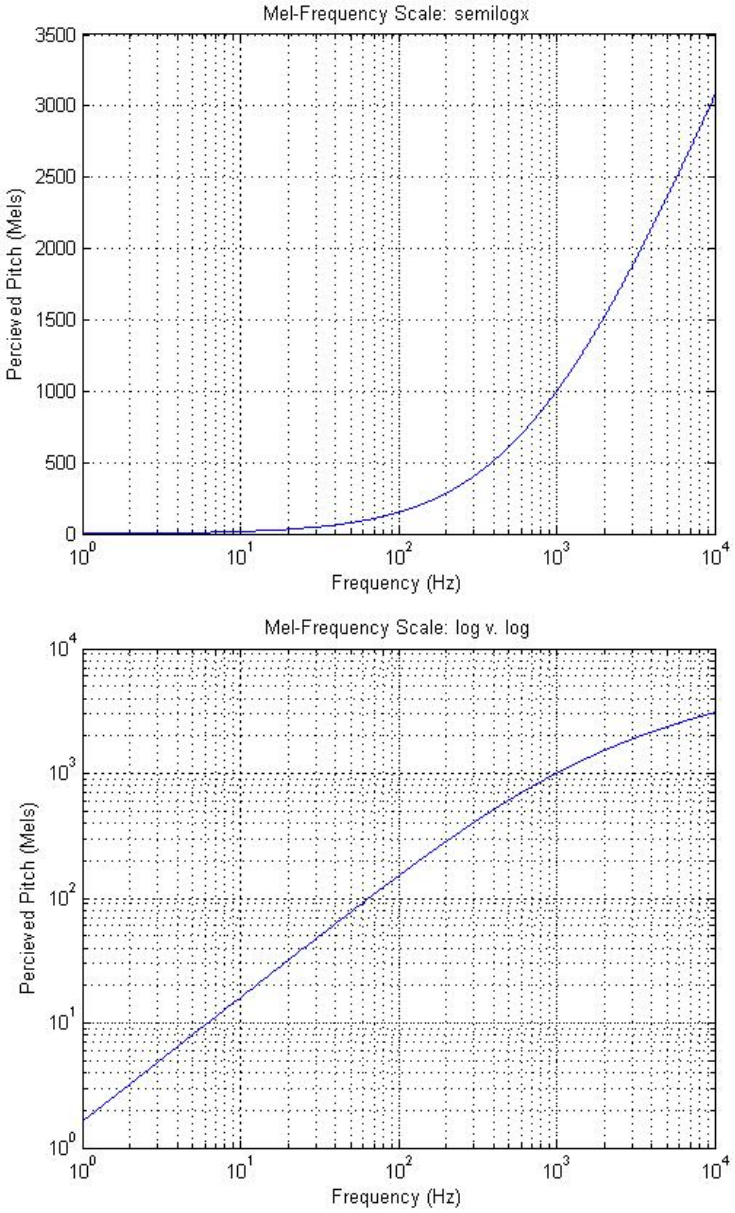

Figure 3.4: Mel-Frequency Scale on Semi-Log (Top) and Log-Log (Bottom) plots.

3.3.1 Mel Frequency Cepstral Coefficients (MF-CC's)

Often, to create feature vectors for an audio signal, a Mel-Scale based filter-bank is used to transform the Fourier spectrum of the signal. After this spectral warping procedure, cepstral coefficients are computed from the transformed spectrum[22]. The filter-bank used is commonly a set of triangular shaped filters, each with unity area, centered on corresponding Mel-frequency indicies.

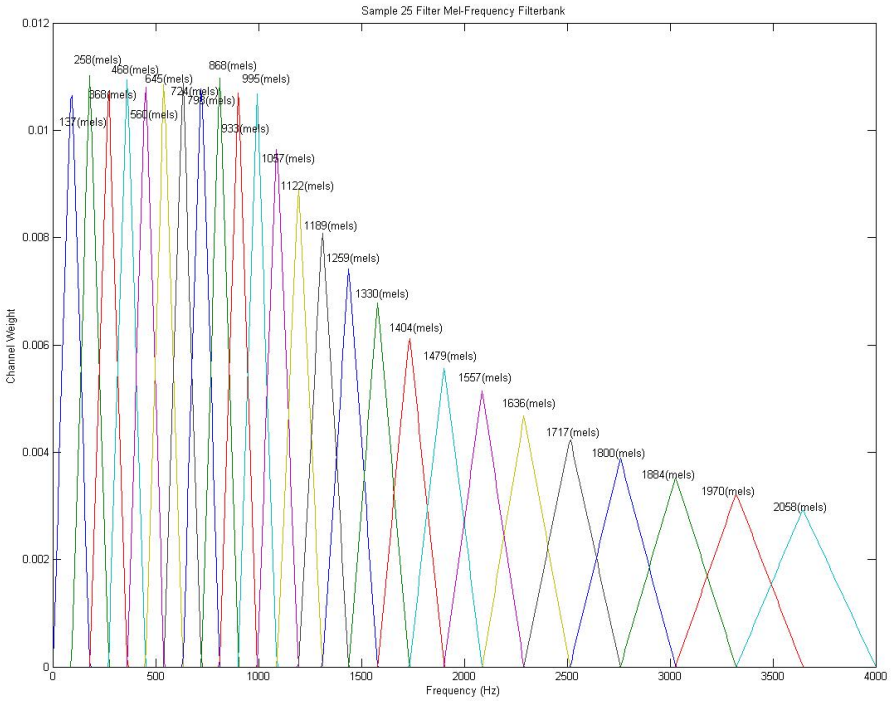

Figure 3.5: Mel-Frequency Filter-bank

The processing steps required to calculate Mel Frequency Cepstral Coefficients are as follows: The absolute value of the Fourier Spectrum of the signal $x(n)$ is squared to give the power spectrum.

$$X_P(k) = |\mathcal{F}[x(n)]|^2$$

The energy in each of the J channels of the filter-bank is then calculated by multiplying each set of channel weights $\phi_j(k)$with $X_P(k)$ and then summing.

$$E_j = \sum_{k=0}^{K-1} \phi_j(k) X_P(k); \;\; 0 \leq j < J$$

Calculating the Cepstral Coefficients is then performed via the inverse Discrete Cosine Transform of the log_{10} of the channel energy.

$$c_m = \sum_{j=0}^{J-1} w_j \, log_{10}(E_j) \, cos\left[\frac{\pi}{J}\left(m - \frac{1}{2}\right) j\right] \;\; m = 1, ..., J$$

$$w_j = \left\{ \begin{array}{cc} \frac{1}{\sqrt{J}}, & j = 0 \\ \sqrt{\frac{2}{J}}, & 1 \leq j < J \end{array} \right\}$$

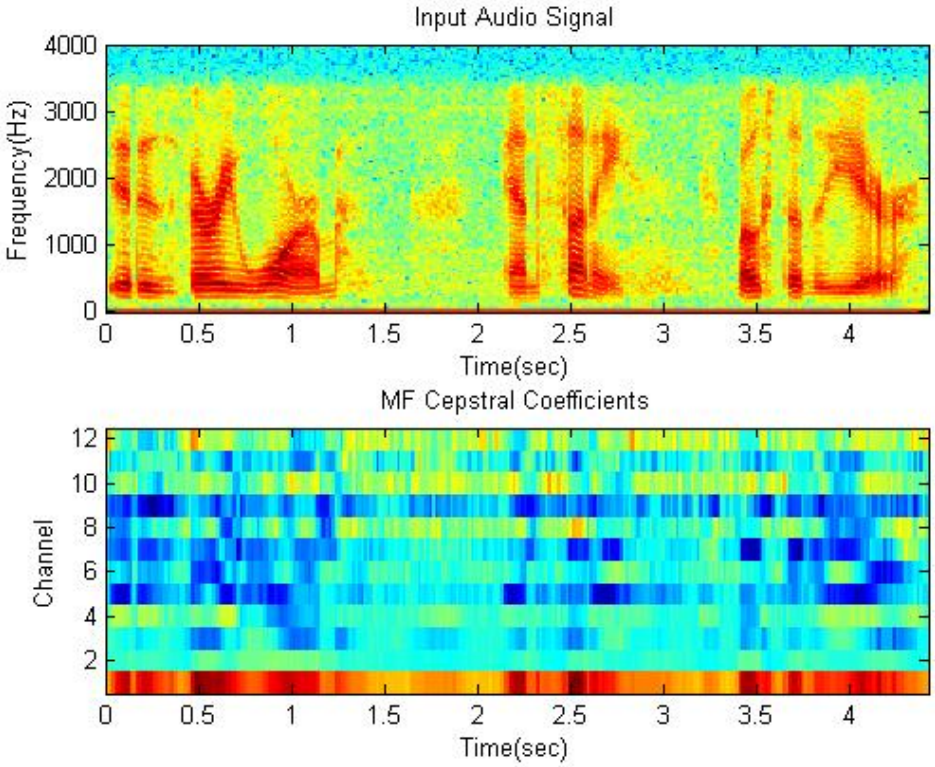

Figure 3.6: Fourier Spectra and Mel Frequency Cepstral Coefficients for an input speech sample. (Top) FFT derived Spectra. (Bottom) 12 point Mel Frequency Cepstral Coefficients with linear weighting.

The process for calculating Mel-Frequency Cepstral Coefficients is implemented in the *melfcc()* function of the RASTA-MAT package.

3.4 Linear Predictive Coding

Linear Predictive Coding (LPC) is a technique that allows for both analysis and synthesis of speech signals by modeling the formant envelope as an all-pole filter.

$$S(z) = E(z)\frac{1}{A(z)}\,(Synthesis)$$

$$E(z) = S(z)A(z)\,(Analysis)$$

where $S(z)$ is the z-transform of the speech waveform, $E(z)$ is the z-transform of the excitation (or LP error) signal, and $A(z)$ is the all-pole filter due to the shape of the acoustic tube, and is defined as

$$A(z) = \sum_{i=0}^{M} a_i z^{-i}\,(a_0 = 1)$$

$$= a_0 + a_1 z^{-1} + ... + a_M z^{-M}$$

having $M + 1$ coefficients.

A linear predictive model of order M will be able to define K possible formant envelope peaks, called formant frequencies, with $M \geq 2K + 1$.[13] When dealing with discrete data the equation for the excitation signal, modeled as an M^{th} order filter approximation, can then be written as:

$$\varepsilon(n) = \sum_{i=0}^{M} a_i\, s(n-i) = s(n) + \sum_{i=1}^{M} a_i\, s(n-i)$$

where $s(n)$ is the discrete-time speech signal, $\varepsilon(n)$ the excitation signal, and $[a_0, ...a_i, ..., a_M]$ being the set of filter coefficients. By further extrapolating this equation we can write the LP error signal as a difference between the actual observed signal $s(n)$ and $\tilde{s}(n)$, which is a prediction signal based on a linear combination of the previous M samples.[13]

$$\varepsilon(n) = s(n) - \tilde{s}(n)$$

$$\tilde{s}(n) = -\sum_{i=1}^{M} a_i \, s(n-i)$$

The Linear Predictive coefficients can be computed using the auto-correlation coefficients by solving a system of linear equations.[15, 16, 11]

$$\begin{bmatrix} R_1 & R_2^* & \cdots & R_P^* \\ R_2 & R_1 & \cdots & R_{P-1}^* \\ \vdots & \ddots & \ddots & \vdots \\ R_P & \cdots & R_2 & R_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_3 \\ \vdots \\ a_{P+1} \end{bmatrix} = \begin{bmatrix} -R_2 \\ -R_3 \\ \vdots \\ -R_{P+1} \end{bmatrix}$$

$$OR$$

$$\begin{bmatrix} R_1 & R_2^* & \cdots & R_P^* \\ R_2 & R_1 & \cdots & R_{P-1}^* \\ \vdots & \ddots & \ddots & \vdots \\ R_P & \cdots & R_2 & R_1 \end{bmatrix}^{-1} \begin{bmatrix} -R_2 \\ -R_3 \\ \vdots \\ -R_{P+1} \end{bmatrix} = \begin{bmatrix} a_2 \\ a_3 \\ \vdots \\ a_{P+1} \end{bmatrix}$$

where $R = [R_1, R_2, \ldots, R_{P+1}]$ is the auto-correlation vector, $a = [a_1, a_2, \ldots, a_{P+1}]$ is the Linear Predictive coefficient vector, P denotes the model order, $[\cdots]^{-1}$ denotes the matrix inverse, and $*$ denotes the complex conjugate operation. In MATLAB the matrix division ('\') operator can be used to perform this operation, however faster algorithms, such as the Levinson-Durbin Recursion Algorithm, for solving the system are included in the *Signal Processing Toolbox* for MATLAB. It should be noted however, that the Levinson Method, while computationally quicker, is historically considered to be less stable than using the matrix inverse.[26]

3.4.1 Linear Predictive Cepstral Coefficients (LP-CC's)

A straightforward method for computing the cepstral coefficients from the linear predictive coefficients is to first convert the LPC coefficients into an N-point Frequency Spectrum by evaluating $H(z) = \frac{1}{A(z)}$, $for\, z = e^{j\,2\pi f}$ at a given set of frequencies f. This can be accomplished using the *freqz()* function in the MATLAB *Signal Processing Toolbox*. Finally, use the Fast Fourier Transform to find the Cepstral Coefficients as described in sec. 2.2.5.

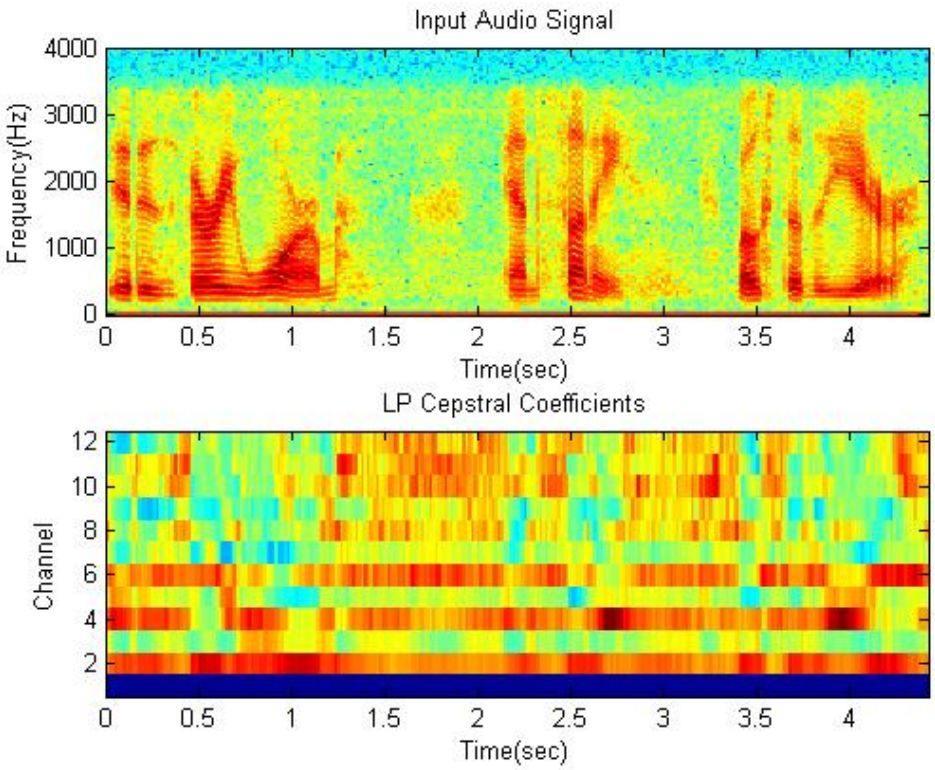

Figure 3.7: Fourier Spectra and LP Cepstral Coefficients for an input speech sample. (Top) FFT derived Spectra. (Bottom) 12 point LPC derived Cepstral Coefficients with linear weighting.

3.4.2 Perceptual Linear Predictive Cepstral Coefficients (PLP-CC's)

Perceptual Linear Prediction of cepstral coefficients combines psycho-acoustic frequency scaling with linear prediction. Hermansky showed that low order PLP analysis could be utilized for improved speaker independence in speech algorithms[10].

In order to calculate Perceptual Linear Prediction coefficients, one needs to first apply psycho-acoustic (Mel, etc.) scaling to the speech spectrum, then use linear prediction techniques to fit an all-pole filter to the re-scaled speech signal spectrum. Finally, cepstral coefficients can be computed from the Perceptual Linear Predictive coefficients as described above. This process is implemented in the *rastaplp()* function of the RASTA-MAT package.

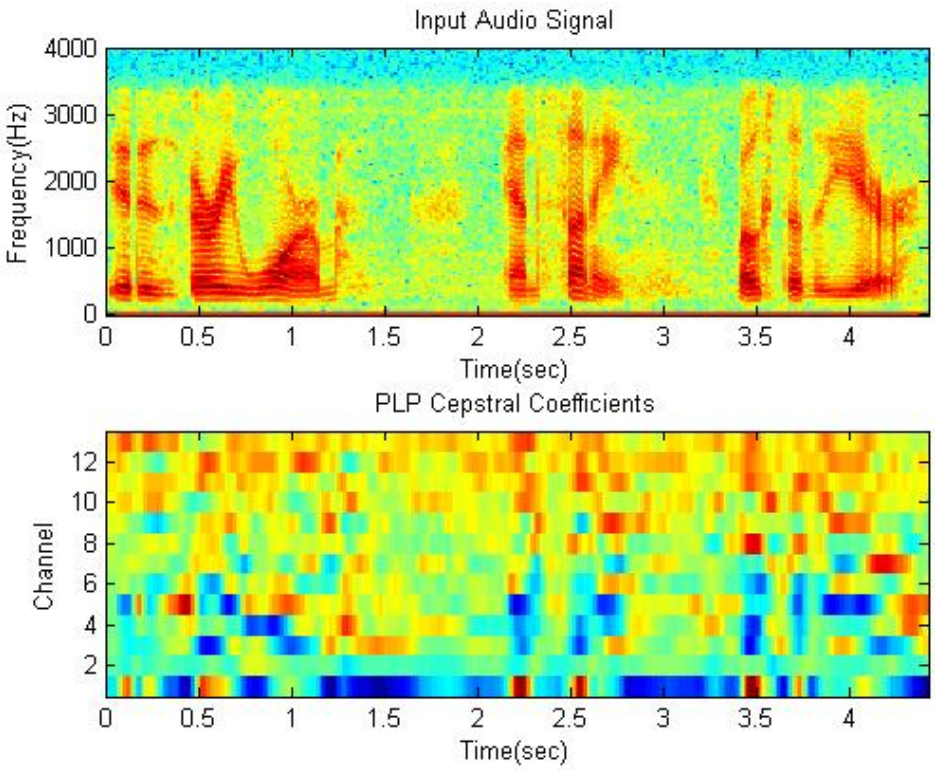

Figure 3.8: Fourier Spectra and PLP Cepstral Coefficients for an input speech sample. (Top) FFT derived Spectra. (Bottom) PLP derived Cepstral Coefficients with linear weighting.

3.5 Cepstral Mean Subtraction

In an effort to mitigate channel effects resulting from telephone transmission lines, cepstral mean subtraction is employed as a feature processing step for all of the speech utterances. Once the cepstral features are calculated using MF / LP / PLP, the mean feature vector for the entire utterance is subtracted off from all of the feature vectors.

3.6 Shifting Delta Operation

The use of the Shifted Delta Cepstral Feature Vectors allows for a pseudo-prosodic feature vector to be computed without having to explicitly find or model the prosodic structure of the speech signal. A shifting delta operation is applied to frame based acoustic feature vectors in order to create the new combined feature vectors for each frame. [2, 14]

3.6.1 Shifted Delta Cepstral Coefficients (SD-MF-CC's, SD-LP-CC's, & SD-PLP-CC's)

The computation of the Shifted Delta feature vectors is a relatively simple procedure. The process is as follows:

The MF, LP, or PLP feature vectors are first computed as described above. Then,

- Let D be the delta distance between acoustic feature vectors
- P be the distance between blocks, and
- K be the number of consecutive blocks used to construct a shifted delta feature vector.

Acoustic feature vectors spaced D sample frames apart are first differenced. Then K differenced feature vector frames, spaced P frames apart, are then stacked to form a new feature vector. Figure 3.9 gives a graphical depiction of this process.

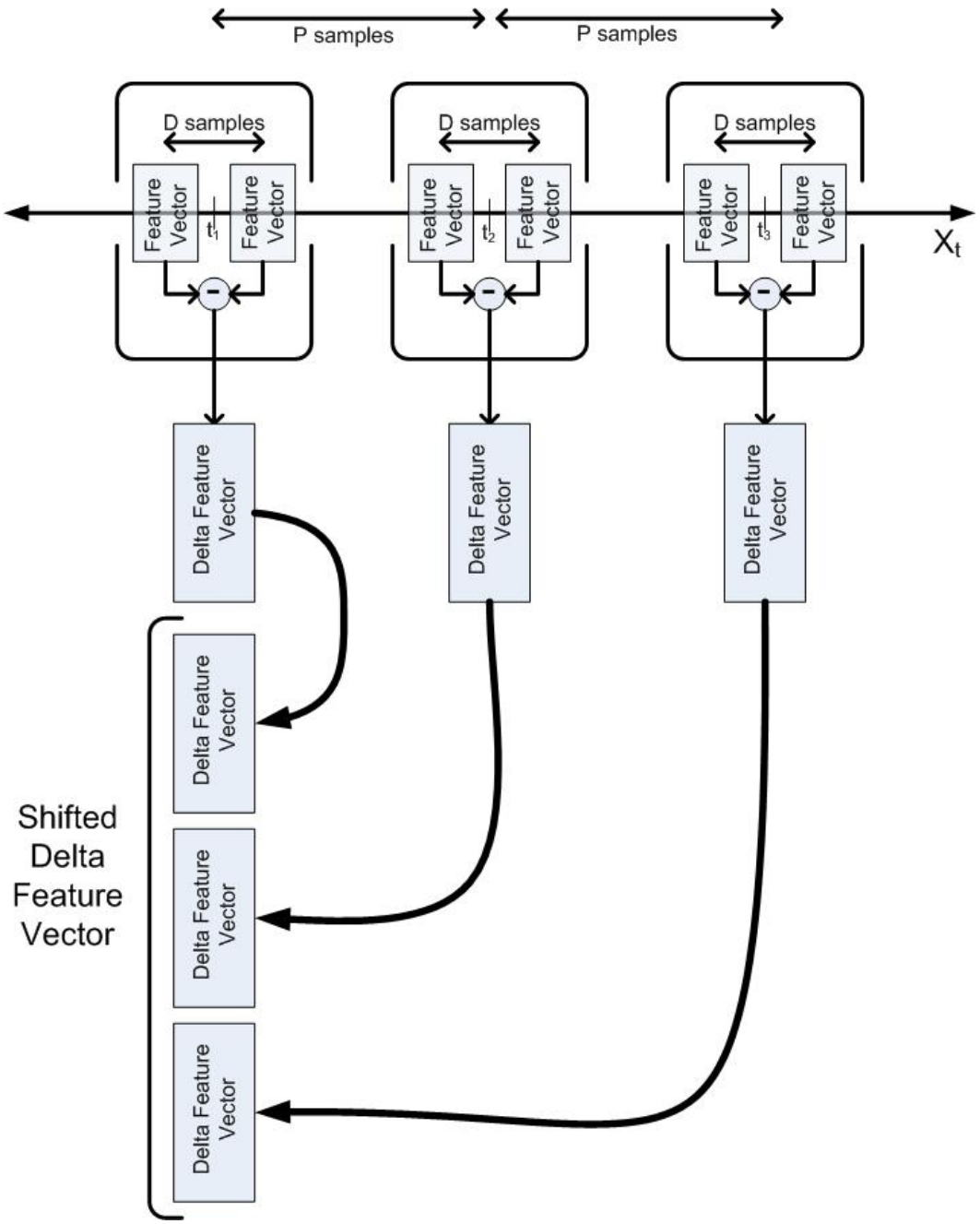

Figure 3.9: Calculation of the Shifted Delta Feature Vectors.

3.7 Silence Removal

A threshold-power based speech/non-speech detection block is then used to remove non-speech frames from the feature vectors. For a given frame of data x, the average power of its Fourier

spectrum is calculated as follows

$$P_{Avg}(x) = \frac{1}{N}\left[\sum_{n=1}^{N} |\mathcal{F}_n(x)|^2\right]$$

where $\mathcal{F}_n(x)$ here refers to the n^{th} term of the Fourier transform of x, and N is the size of the Fourier Transform vector.

A threshold value T is used to determine if the frame is removed or not. If $P_{Avg}(x) < T$, the frame x is removed from the set of feature vectors[1]. The default threshold level is $T = .00056234$, and is equivalent to -65dB. A Threshold level of 0 (which is equivalent to $-\infty$ in dB) results in no frames being removed from the set of feature vectors.

[1] The silence removal utility function can also operate as a downward expanding noise gate that first marks indicies that are above the threshold value, and then uses a set of forward and backward passes to expand these peaks out to their nearest local minimum. However, for this thesis only the straightforward thresholding functionality was utilized.

Chapter 4

Gaussian Mixture Models

Gaussian Mixture Models serve as methods to describe complex N-dimensional distributions of data points in a feature space. Much like how Fourier analysis uses additive sinusoids to describe a signal, Gaussian Mixture Models use combinations of multivariate Gaussian distributions to summarize the entire distribution over the feature space. Gaussian Mixture Models are a semi-parametric technique for estimating probability density functions from labeled or unlabeled data. [18]

4.1 The Multivariate Gaussian (or Normal) Distribution

The multivariate normal probability density is:

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}} |\sum|^{\frac{1}{2}}} e^{\left\{-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)\right\}}$$

where x is a n-dimensional vector of random variables, μ is the mean vector of the probability distribution, and $\sum$ is an [n by n] covariance matrix[1], with determinant $|\sum|$. $\sum^{-1}$ denotes the covariance matrix inverse (also known as the precision matrix). The superscript ^{T} denotes the transpose operation: row vectors become column vectors and visa versa.

[1] While using the $\sum$ symbol to denote the covariance matrix in this manner may be a little confusing at first for those unfamiliar with multivariate statistics notation, the practice is quite common in the literature, and so we continue with it here. Nonetheless, care should be taken in observing the context of the symbol's use, as to weather it is implying the customary mathematical summation operation, or a covariance matrix variable. An easy way to avoid confusion is to look for limits of summation above and below the $\sum$ symbol, thus denoting the summation operation.

4.2 Mixture Models

A mixture model for estimating a probability density function using M multivariate Gaussian distributions as basis functions can be written as:

$$p(x) = \sum_{j=1}^{M} p(x|j)w(j)$$

with the constraints that

$$\sum_{j=1}^{M} w(j) = 1$$

$$0 \leq w(j) \leq 1$$

$$\int p(x|j)dx = 1$$

where $w(j)$ is the mixture weight (or prior probability) associated with the mixture component j, and $p(x|j)$ is the multivariate Gaussian distribution for the j^{th} mixture component. [1] Training of Gaussian Mixture Models is usually accomplished through the use of k-means clustering for initialization, along with several iterations of the EM algorithm, both of which are included in the NETLAB software package.

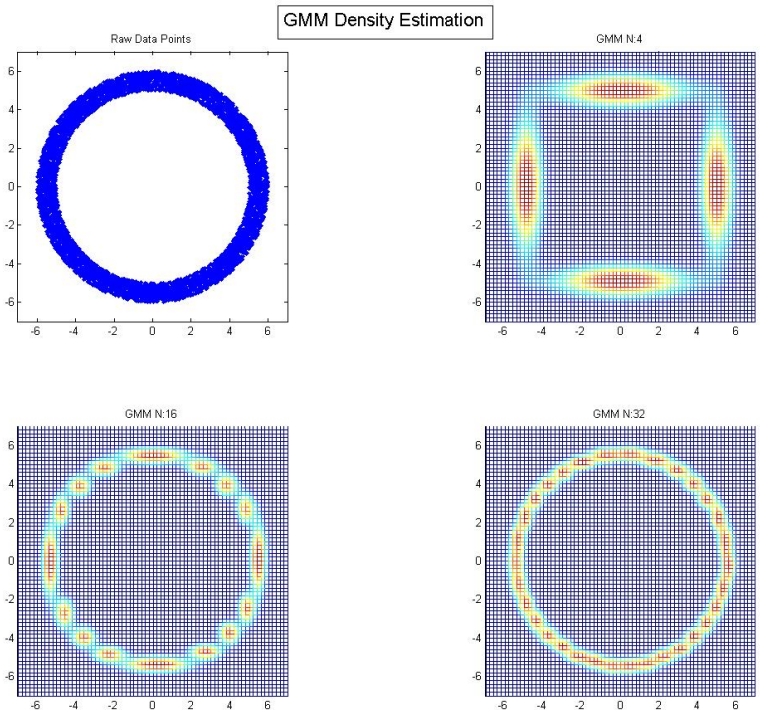

Figure 4.1: GMM Density Estimation Demo.
(Upper Left) 2 Dimensional Raw data points. (Upper Right) GMM Estimation with 4 mixture components. (Lower left) GMM Estimation with 16 mixture components. (Lower Right) GMM Estimation with 32 mixture components. Gaussian Mixture Models are using diagonal covariance matrices.

Because of the quasi-parametric nature of GMM's, arbitrary feature space segmentations can be modeled to very high accuracy given enough training data and enough mixture components. With a GMM, the user is not restricted to specific functional forms, as in truly parametric modeling. Yet the size of the model only grows with the complexity of the problem being solved, unlike fully non-parametric methods.[1] Also, GMM's are capable of modeling the density of data points along an arbitrary feature space segmentation curve. This allows a GMM to discriminate between classes that lie on the same feature space curve, but have different densities along that curve. The acoustic feature vectors of the different phonemes of languages is one example of this. For the most part, the phonemes of the languages lie within the same bounded surface of the feature space, with the differentiating factor between languages being the distribution of data points along the feature space surface.

While these traits of GMM's are exceptionally noteworthy, one must be careful to ensure that enough training data and mixture components are used to accurately describe the classification boundary in the feature space accurately. The more mixture components used, the more accurate the model can become, as can be seen in the plots presented in section 6.6. But increasing the number of mixtures also increases the amount of training data and time required for processing. Such trade-offs should be considered when using GMMs.

Chapter 5

Method

5.1 System Overview

The flow chart in figure 5.1 depicts the overall organization of the system. The system is built with a modular architecture. Two main modules for training and testing incorporate the use of smaller modules for feature extraction and GMM distance measurements. The system uses default Feature Extraction and GMM Distance Metric functions, but is scalable so that user-supplied MATLAB m-files can be easily plugged in to replace the default functions.

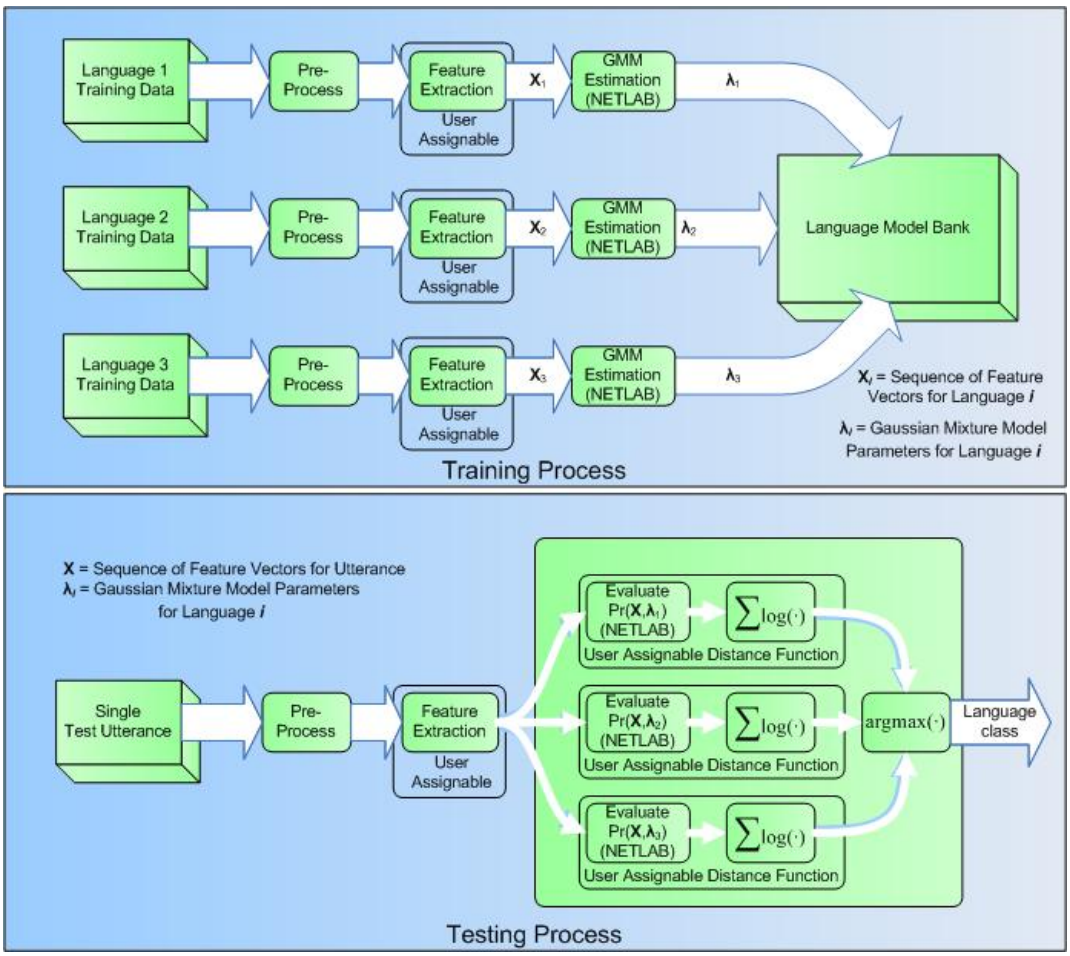

Figure 5.1: Language Identification System Flow Chart

5.2 Software Architecture

This section gives a general description of how each of the main modules are organized and used. Full source code of each of the software modules can be found in Appendix A, with an example script showing how to run a full training and testing procedure for a set of data in A.5.

5.2.1 Training

Training data is stored in a set of directory paths, one path per language to be identified. The set of directory path strings are passed to the training algorithm as well as language labels for each directory. For each language directory supplied, the training algorithm uses the .wav files

located in the directory and the selected feature extraction function to develop and train a GMM representative of that language. The function prototype and argument list is as follows:

```
function [GMMLangs] = JL_LID_Train(TrainDirs,LANGUAGES,...
    MAXFS,NORM_FEATS,func,funcArgs,SecondsOfSpeech,NumMixtures,ItrEM)
% Returns a cell array of Gaussian Mixture Models for each training set.
% <TrainDirs>   Cell array of directory strings.  For each directory, a GMM
%               will be trained using all of the .WAV files in that
%               directory.
% <LANGUAGES>   Cell array of descriptive strings that label each directory
% <MAXFS>       Maximum Sampling Frequency
% <NORM_FEATS>  [0|1] Whiten the feature vectors before training
% <func>        Function pointer to the feature vector extraction function.
% <funcArgs>    Arguments for the feature vector extraction function.
% <SecondsOfSpeech>    Amount of speech (s) to use for training. A value of
%               -1 uses all available frames.  Otherwise, if the
%               cumulative amount of frames returned by the feature
%               extraction function multiplied by the seconds represented
%               by each frame is greater than SecondsOfSpeech, frames of
%               feature vectors are randomly selected so that the number of
%               retained frames multiplied by the seconds represented
%               by each frame is aproximately equal to SecondsOfSpeech.
%               The retained frames are then used to train the GMM.
% <NumMixtures> Number of mixtures to be used for each GMM.
% <ItrEM>       Number of Iterations of EM algorithm to use.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com
```

5.2.2 Testing

In a manner similar to the training module, the testing module accepts a list of directory path strings in addition to a set of Gaussian Mixture Models and a corresponding set of language labels for each model. For each directory supplied, the module computes the feature vectors for each .wav file, and tests each .wav file's feature vector's distance from each GMM supplied. The module outputs a confusion matrix (one row per directory, and one column per GMM), along with the number of files tested in each directory, and a list of each tested file-name with its detected language label. The function prototype and argument list is as follows:

```
function [confMat,nFiles,fileList] = JL_LID_Test(varargin)
%Test each of the .wav files in the directories entered against each of the
%stored GMM's for each LANGUAGE.  Arguments are string-value pairs.
% <TestDirs>    Cell array of directory strings.  For each directory, each
```

```
%               .wav file in the directory will be tested against each of
%               the GMM's.
% <Languages>   Cell array of descriptive strings that label each GMM.
% <FeatFunc>    Function pointer to the feature vector extraction function.
% <FeatFuncArgs>Arguments for the feature vector extraction function.
% <GMMs>        The Gaussian Mixture Models for each language.
%
%These additional arguments are optional and/or have default preset values:
% <MaxFS>       (8000) Maximum Sampling Frequency
% <NormFeats>   [0|(1)] Whiten the feature vectors before training
% <GMMDistMode> Operating mode of the GMM distance function.('PROB')
% <GMMDistFunc> Function pointer for the distance function used to evaluate
%               the feature vectors on each GMM.  (@gmmdist)
% <GMMDistFuncArgs> Additional Arguments for the GMM distance function.([])
% <GMM-UBM>      (optional) Universal Background Model to use for distance
%               evaluation.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com
```

5.2.3 Feature Extraction

The feature extraction function prototype accepts as input a mono audio signal vector, sampling frequency, and a list of string-value parameter arguments. It returns a set of multi-dimensional feature vectors for the audio signal vector in matrix form, as well as the cumulative amount of speech represented by the feature vector matrix (in seconds), and the amount of speech (in seconds) that each feature vector represents. Rows of the feature vector matrix represent the different feature dimensions, and each column is a unique feature vector. The default function prototype and argument list is as follows:

```
function [feat,SecondsOfSpeech,SecondsPerFrame] = JL_GET_FEATS(mix,fs,varargin)
%  Gets the feature vectors for the signal MIX with sampling frequency FS
%  and parameters supplied by the set of string-value pairs given in
%  VARARGIN.
%
%Parameters: Brackets denote an optional parameter.  Parenthesese denote
%           the default setting for that parameter.
%'print'            [(0)|1]  Display incremental output.
%'nfft'             (256) Size of FFT to use when calculating features
%'win'              (256) Size of Window to use when calculating features
%'ov'               (128) Amount of Overlap between calculation windows
%'minHz'            (300) Minimum frequency to allow.  300Hz is the
%                      standard cutoff frequency for telephone speech high
%                      pass filter.
%'numCoeff'         (12) Number of Cepstral Coefficients to use
%'numCoeffLP'       (12) Number of LP Coefficients to use when calculating
```

```
%                       LP-CC features.
%'PRE_EMPH'         [(0)|1] Use Pre-Emphasis Filtering to adjust for
%                       natural ~20dB/decade rolloff of the human voice.
%'ENHANCE'          [0|(1)] Use Cepstral Speech Enhancement algorithm as a
%                       pre-processor.
%'USE_CMS'          [0|(1)] Use Cepstral Mean Subtraction to try to
%                       mitigate channel effects.
%'USE_SDC'          [0|(1)] Use Shifted Delta Cepstral Coefficients
%'USE_DELTA'        [(0)|1] Use Delta Coefficients
%'USE_POWER_TERM'   [(0)|1] Use or omit the power (first) cepstral term
%'deltaDist'        ([]) The distance (in feature frames) to use when
%                       calculating delta coefficients.
%'SDC_Block_Spacing' ([]) The distance (in frames) to use between shifted
%                       delta blocks.
%'deltaDist_Sec'    (.1920) The distance (in seconds) to use when
%                       calculating delta coefficients.
%'SDC_Block_Spacing_Sec' (.048) The distance (in seconds) to use between
%                       shifted delta blocks.
%'SDC_Blocks'       (3) The Number of Shifted Delta Blocks to Use
%'LifterExp'        (0) The exponent to use when liftering (i.e. weighting)
%                       the cepstral coefficients
%'Mode'             ['LP-CC'|'PLP-CC'|'MF-CC'|'CUST'] Feature Calculation
%                       mode to use.  'LP-CC' - Linear Prediction derived
%                       Cepstral Coefficients.  'PLP-CC' - Perceptual
%                       Linear Prediction Cepstral Coefficients.  'MF-CC' -
%                       Mel Frequency Cepstral Coefficients.  'CUST' -
%                       Custom cepstral coefficient derivation function.
%                       Use of the 'CUST' option means one must also define
%                       'CustModeFunction' and 'CustModeFunctionArgs'.
%'CustModeFunction' ([]) A function handle to a user defined function for
%                       calculating the cepstral coefficients that obey the
%                       following prototype:
%                           CepCoeffs = FUNC(mix,fs,args)
%'CustModeFunctionArgs' ([]) Supplimental arguments for the custom mode
%                       function.
%'VTHRESH'          (.00056234) Threshold for determining if a
%                       frame is speech/non-speech data. .00056234 is
%                       equivalent to -65dB.
%'V_Max'            [(0)|1] Only use the locations of peaks in the
%                       sequential power signature of feature frames to
%                       extract speech frames.  Greatly reduced the number
%                       of feature frames extracted, but helps to ensure
%                       that only features frames corresponding to actual
%                       speech data are used.  Was implemented for
%                       debugging purposes only.  Not Recommended.
%'VOP'              [(0)|1] Try to use the locations of Vocal Onset Points
%                       to determine which frames are extracted from te
%                       audio signal. Again, was implemented as a debugging
%                       tool and is not recommended.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com
```

The default supplied feature extraction function first pre-processes the incoming audio according to the parameters provided, then computes LP, MF, or PLP cepstral coefficients for the audio vector. If shifted delta features are desired, the module then computes these as described in 3.9.

5.2.4 GMM Distance Metric

The GMM distance function accepts a Gaussian Mixture Model and a set of feature vectors and returns a numerical metric for how 'far' each feature vector is away from the GMM. The function prototype and argument list is as follows:

```
function d = gmmdist(GMM1,feat,mode,GMM2,varargin);
%Default GMM distance calculation function. For each feature vector the
%function returns a distnace <d>.  Custom distance functions must follow
%the same parameter passing scheme.
%<GMM1>     Primary Gaussian Mixture Model used to evaluate each vector in
%           <feat>.
%<feat>     Set of feature vectors to evaluate.  [N by M] with N being the
%           number of feature vectors and M being each vector length.
%<mode>     [('PROB') | 'Sym-KL' | 'KL']
%           - 'PROB' - Uses the probability of each feature vector
%           falling on GMM1.  This is the most basic and straightforward
%           distance metric. GMM2 is ignored in this mode.
%           - 'KL' - Uses the Kullback-Liebler Divergence to calculate the
%           asymetric distance of the features between GMM1 and GMM2.
%           - 'Sym-KL' - Uses the Kullback-Liebler Divergence to calculate
%           the symetric distance of the features between GMM1 and GMM2.
%<GMM2>     Secondary Gaussian Mizture Model used for KL distance modes.
%<varargin> Is ignored in this default distance function.  Intended so that
%           custom distance functions can pass additional parameters if
%           needed.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com
```

The *mode* parameter allows for selectable operation, and an additional Gaussian Mixture Model to compare against can be passed in *GMM2*, if so desired. The *GMM2* parameter is optional and so far unused in the experiments presented here. The default supplied GMM distance function mode ('PROB') uses the log-likelihood values for each feature vector falling on the distribution given by the GMM as calculated by the NETLAB package functions.

The returned value of the GMM distance function can be either a vector (one number per feature vector) or a single numerical answer. If the output is a vector it will be summed by the testing function to form a numerical representation of the distance between the input set of feature vectors

and the supplied GMM. The syntax for calling the GMM Distance function used in the testing routine is as follows:

```
...Miscellaneous Code...
c(i) = sum(GMMDistFunc(GMM1,feat',GMMDistMode,UBM,GMMDistFuncArgs));
...Miscellaneous Code...
```

Chapter 6

Experiments

This section presents and discusses the experimental data that was collected on the GMM Language Identification task.

6.0.5 Covariance Matrix Type

For the experiments in Language Identification, it was decided to use diagonal co-variance matrices for the Gaussian Mixture Models in order to avoid memory issues, and to decrease computation time as compared to using full co-variance matrices. This has the effect of only allowing the Gaussian Mixture variances to align along the feature dimension axes, as the co-variance between each feature dimension is kept to zero.

6.0.6 Number of Mixtures

64 was chosen to be the number of mixtures used for the Gaussian Mixture Models for the majority of experiments for a number of heuristic reasons.

- It gives a reasonable and consistent estimate for the number of phonemes that can be expected in any given language.
- Keeping the number of mixtures used low allows for faster algorithm speed and GMM training.
- Reduces memory constrains on the system as compared to using higher numbers of mixture components.
- Requires less training data to obtain accurate results than using higher mixture orders.

Section 6.5 experimentally addresses this issue.

6.0.7 Amount of Training Data per Language

In training multiple GMM's it is important to be consist ant with the amount of training data used for training. For the majority of the experiments conducted, 1800 seconds worth of training feature vectors for each language to be modeled were randomly selected as the training set. Section 6.5 experimentally addresses this issue.

6.0.8 Training and Testing Data Sets

The testing and training sets for all of the experiments presented here were mutually exclusive, and approximately equal size, subsets created from the 'STB' type OGI Database files. The 'STB' type denotes free speech samples with a maximum duration of 50 seconds.

6.1 Telephone Speech Language Identification Task Results by Feature Type - CMS,CSE, & PE Enabled

For this set of experiments the software parameters were as follows:

- Cepstral Mean Subtraction: Enabled
- Cepstral Speech Enhancement: Enabled
- Pre-Emphasis: Enabled
- Number of Mixtures: 64
- Amount of Training Speech Used to train each GMM: 1800 seconds
- GMM EM iterations: 10
- Voicing Threshold: -65dB
- Number of Cepstral Coefficients Calculated: 12
- NFFT: 256 Point
- Window Size: 256 samples

- Window Type: Hamming
- Window Overlap: 128 Samples
- Number of LPC/PLP Coefficients Calculated (When Applicable): 12
- Delta Distance (When Applicable): 0.192 seconds
- Shifted Delta Spacing (When Applicable): 0.048 seconds
- Number of Shifted Deltas (When Applicable): 3

The Experiments were programmed in MATLAB 7 and compiled and run on 9/24/06 on a Dual Core Intel Pentium(R) 4 CPU 3.00GHz, with 2GB of RAM, Microsoft Windows XP Professional SP2. The results are tabulated below.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	47.06%	7.84%	45.10%		
English	7.69%	76.92%	15.38%		
Japanese	40.00%	6.67%	53.33%		
Average				***59.11%***	
STD Deviation					***15.75%***

Table 6.1: Results for LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	60.78%	11.76%	27.45%		
English	6.59%	87.91%	5.49%		
Japanese	24.44%	11.11%	64.44%		
Average				***71.05%***	
STD Deviation					***14.72%***

Table 6.2: Results for SD-LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	37.25%	15.69%	47.06%		
English	13.19%	70.33%	16.48%		
Japanese	33.33%	13.33%	53.33%		
Average				***53.64%***	
STD Deviation					***16.54%***

Table 6.3: Results for MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	49.02%	25.49%	25.49%		
English	9.89%	89.01%	1.10%		
Japanese	13.33%	20.00%	66.67%		
Average				***68.23%***	
STD Deviation					***20.04%***

Table 6.4: Results for SD-MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	39.22%	19.61%	41.18%		
English	8.79%	75.82%	15.38%		
Japanese	26.67%	15.56%	57.78%		
Average				***57.61%***	
STD Deviation					***18.30%***

Table 6.5: Results for PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	56.86%	25.49%	17.65%		
English	12.09%	84.62%	3.30%		
Japanese	24.44%	15.56%	60.00%		
Average				***67.16%***	
STD Deviation					***15.20%***

Table 6.6: Results for SD-PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

6.2 Telephone Speech Language Identification Task Results by Feature Type - CMS,CSE, & PE Disabled

For this set of experiments the software parameters were as follows:

- Cepstral Mean Subtraction: Disabled

- Cepstral Speech Enhancement: Disabled
- Pre-Emphasis: Disabled
- Number of Mixtures: 64
- Amount of Training Speech Used to train each GMM: 1800 seconds
- GMM EM iterations: 10
- Voicing Threshold: -65dB
- Number of Cepstral Coefficients Calculated: 12
- NFFT: 256 Point
- Window Size: 256 samples
- Window Type: Hamming
- Window Overlap: 128 Samples
- Number of LPC/PLP Coefficients Calculated (When Applicable): 12
- Delta Distance (When Applicable): 0.192 seconds
- Shifted Delta Spacing (When Applicable): 0.048 seconds
- Number of Shifted Deltas (When Applicable): 3

The Experiments were programmed in MATLAB 7 and compiled and run on 9/22/06 a Dual Core Intel Pentium(R) 4 CPU 3.00GHz, with 2GB of RAM, Microsoft Windows XP Professional SP2. The results are tabulated below.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	35.29%	27.45%	37.25%		
English	21.98%	64.84%	13.19%		
Japanese	24.44%	33.33%	42.22%		
Average				***47.45%***	
STD Deviation					***15.45%***

Table 6.7: Results for LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	52.94%	27.45%	19.61%		
English	12.09%	83.56%	4.40%		
Japanese	22.22%	17.78%	60.00%		
Average				***65.49%***	
STD Deviation					***16.01%***

Table 6.8: Results for SD-LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	37.25%	27.45%	35.29%		
English	9.89%	78.02%	12.09%		
Japanese	13.33%	44.44%	42.22%		
Average				***52.50%***	
STD Deviation					***22.24%***

Table 6.9: Results for MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	52.94%	29.41%	17.65%		
English	12.09%	85.71%	2.20%		
Japanese	26.67%	22.22%	51.11%		
Average				***63.26%***	
STD Deviation					***19.47%***

Table 6.10: Results for SD-MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	49.02%	19.61%	31.37%		
English	31.87%	49.45%	18.68%		
Japanese	24.44%	31.11%	44.44%		
Average				***47.64%***	
STD Deviation					***2.77%***

Table 6.11: Results for PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	58.82%	19.61%	21.57%		
English	23.08%	69.23%	7.69%		
Japanese	31.11%	13.33%	55.56%		
Average				***61.20%***	
STD Deviation					***7.14%***

Table 6.12: Results for SD-PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

6.3 Telephone Speech Language Identification Task Results by Feature Type - w/ 128 GMM Mixtures & 3600 Seconds Training Data per GMM

For this set of experiments the software parameters were as follows:

- Cepstral Mean Subtraction: Enabled
- Cepstral Speech Enhancement: Enabled
- Pre-Emphasis: Enabled
- Number of Mixtures: 128
- Amount of Training Speech Used to train each GMM: 3600 seconds
- GMM EM iterations: 10
- Voicing Threshold: -65dB
- Number of Cepstral Coefficients Calculated: 12
- NFFT: 256 Point
- Window Size: 256 samples
- Window Type: Hamming
- Window Overlap: 200 Samples
- Number of LPC/PLP Coefficients Calculated (When Applicable): 12

- Delta Distance (When Applicable): 0.192 seconds
- Shifted Delta Spacing (When Applicable): 0.048 seconds
- Number of Shifted Deltas (When Applicable): 3

The Experiments were programmed in MATLAB 7 and compiled and run on 9/15/2006 on a Dual Core Intel Pentium(R) 4 CPU 3.00GHz, with 2GB of RAM, Microsoft Windows XP Professional SP2. The results are tabulated below.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	49.02%	7.84%	43.14%		
English	14.29%	73.63%	12.09%		
Japanese	37.78%	6.67%	55.56%		
Average				***59.40%***	
STD Deviation					***12.75%***

Table 6.13: Results for LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	64.71%	11.76%	23.53%		
English	2.20%	93.41%	4.40%		
Japanese	31.11%	11.11%	57.78%		
Average				***71.96***%	
STD Deviation					***18.89%***

Table 6.14: Results for SD-LP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	43.14%	11.76%	45.10%		
English	15.38%	70.33%	14.29%		
Japanese	35.56%	13.33%	51.11%		
Average				***54.86***%	
STD Deviation					***13.98%***

Table 6.15: Results for MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	47.06%	31.37%	21.57%		
English	3.30%	96.70%	0.0%		
Japanese	17.78%	22.22%	60.00%		
Average				***67.92%***	
STD Deviation					***25.75%***

Table 6.16: Results for SD-MF-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	50.98%	13.73%	35.29%		
English	6.59%	80.22%	13.19%		
Japanese	26.67%	20.00%	53.33%		
Average				***61.51%***	
STD Deviation					***16.24%***

Table 6.17: Results for PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix	German	English	Japanese	Average	STD Deviation
German	43.14%	29.41%	27.45%		
English	9.89%	89.01%	1.10%		
Japanese	24.44%	17.78%	57.78%		
Average				***63.31%***	
STD Deviation					***23.43%***

Table 6.18: Results for SD-PLP-CC Features.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

6.4 Telephone Speech 10 language LID Task Results by Feature Type

Although this work's primary focus is on a 3 Language task, an experiment analyzing the full 10-language performance of the algorithm was conducted, in an effort to examine the scalability of the algorithm, and to see if the observed superiority of SD-LP-CC features translates accordingly.

For this set of experiments the software parameters were as follows:

- Cepstral Mean Subtraction: Enabled
- Cepstral Speech Enhancement: Enabled
- Pre-Emphasis: Enabled
- Number of Mixtures: 64
- Amount of Training Speech Used to train each GMM: 1800 seconds
- GMM EM iterations: 10
- Voicing Threshold: -65dB
- Number of Cepstral Coefficients Calculated: 12
- NFFT: 256 Point
- Window Size: 256 samples
- Window Type: Hamming
- Window Overlap: 128 Samples
- Number of LPC/PLP Coefficients Calculated (When Applicable): 12
- Delta Distance (When Applicable): 0.192 seconds
- Shifted Delta Spacing (When Applicable): 0.048 seconds
- Number of Shifted Deltas (When Applicable): 3

The Experiments were programmed in MATLAB 7 and compiled and run on 9/26/06 on a Dual Core Intel Pentium(R) 4 CPU 3.00GHz, with 2GB of RAM, Microsoft Windows XP Professional SP2. The results are tabulated below.

Grayed entries indicate off-diagonal values that are greater than or equal to the diagonal entry for that row. Only the SD-LP-CC feature vector confusion matrix contains no off-diagonal entries greater than the diagonal, and has all diagonal entries greater than statistical chance (10%). The overall average accuracy of the SD-LP-CC 10 Language Task experiment is 47.21%, with the standard deviation along the diagonal of the confusion matrix at 18.81%.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**9.80**	5.88	***11.76***	3.92	***15.69***	***11.76***	***31.37***	3.92	3.92	1.96		
English	2.20	**60.44**	4.40	2.20	3.30	2.20	14.29	5.49	4.40	1.10		
Japanese	6.67	0.00	**13.33**	6.67	6.67	***20.00***	***26.67***	4.44	8.89	6.67		
French	1.92	0.00	3.85	**61.54**	17.31	3.85	9.62	1.92	0.00	0.00		
Farsi	8.89	4.44	4.44	0.00	**51.11**	8.89	13.33	2.22	4.44	2.22		
Korean	2.44	0.00	4.88	0.00	12.20	**36.59**	29.27	0.00	7.32	7.32		
Mandarin	12.00	2.00	4.00	12.00	10.00	10.00	**48.00**	0.00	0.00	2.00		
Spanish	7.41	9.26	7.41	3.70	7.41	9.26	9.26	**42.59**	3.70	0.00		
Tamil	3.77	1.89	1.89	0.00	3.77	9.43	5.66	3.77	**67.92**	1.89		
Vietnam	0.00	0.00	20.00	2.22	8.89	2.22	***22.22***	2.22	20.00	**22.22**		
Average											***41.35***	
STDV												***20.52***

Table 6.19: Results for LP-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**21.57**	7.84	17.65	3.92	15.69	3.92	15.69	3.92	3.92	5.88		
English	1.10	**81.32**	0.00	0.00	1.10	0.00	6.59	2.20	6.59	1.10		
Japanese	4.44	0.00	***26.67***	8.89	6.67	15.56	17.78	15.56	2.22	2.22		
French	0.00	0.00	1.92	**69.23**	7.69	11.54	5.77	1.92	1.92	0.00		
Farsi	0.00	4.44	2.22	6.67	**51.11**	8.89	4.44	8.89	4.44	8.89		
Korean	2.44	4.88	9.76	7.32	0.00	**39.02**	14.63	7.32	7.32	7.32		
Mandarin	4.00	0.00	6.00	14.00	12.00	12.00	**42.00**	6.00	2.00	2.00		
Spanish	0.00	11.11	9.26	11.11	1.85	14.81	1.85	**38.89**	7.41	3.70		
Tamil	3.77	1.89	0.00	7.55	3.77	3.77	1.89	3.77	**62.26**	11.32		
Vietnam	6.67	0.00	0.00	2.22	11.11	2.22	20.00	6.67	11.11	**40.00**		
Average											***47.21***	
STDV												***18.81***

Table 6.20: Results for SD-LP-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**13.73**	0.00	11.76	5.88	11.76	***15.69***	***27.45***	3.92	3.92	5.88		
English	3.30	**60.44**	3.30	3.30	4.40	4.40	12.09	3.30	3.30	2.20		
Japanese	8.89	0.00	**17.78**	4.44	4.44	13.33	***17.78***	13.33	13.33	6.67		
French	0.00	0.00	0.00	**69.23**	11.54	5.77	9.62	3.85	0.00	0.00		
Farsi	2.22	2.22	0.00	2.22	**57.78**	6.67	11.11	2.22	11.11	4.44		
Korean	0.00	0.00	7.32	0.00	7.32	**46.34**	24.39	2.44	4.88	7.32		
Mandarin	0.00	6.00	4.00	12.00	10.00	18.00	**46.00**	2.00	2.00	0.00		
Spanish	7.41	3.70	7.41	11.11	9.26	9.26	14.81	**25.93**	11.11	0.00		
Tamil	1.89	0.00	1.89	0.00	3.77	7.55	5.66	3.77	**71.70**	3.77		
Vietnam	2.22	0.00	6.67	0.00	11.11	0.00	24.44	2.22	22.22	**31.11**		
Average											***44.00***	
STDV												***21.00***

Table 6.21: Results for MF-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**21.57**	11.76	7.84	9.80	3.92	7.84	11.76	9.80	5.88	9.80		
English	1.10	**82.42**	0.00	3.30	1.10	2.20	1.10	5.49	3.30	0.00		
Japanese	2.22	6.67	**17.78**	15.56	2.22	13.33	2.22	**22.22**	6.67	11.11		
French	3.85	0.00	1.92	**63.46**	3.85	11.54	3.85	9.62	0.00	1.92		
Farsi	0.00	4.44	2.22	0.00	**60.00**	6.67	4.44	8.89	4.44	8.89		
Korean	0.00	0.00	4.88	4.88	9.76	**36.59**	9.76	17.07	7.32	9.76		
Mandarin	6.00	6.00	6.00	10.00	14.00	10.00	**32.00**	8.00	0.00	8.00		
Spanish	3.70	9.26	1.85	1.85	11.11	12.96	0.00	**48.15**	7.41	3.70		
Tamil	0.00	1.89	0.00	1.89	5.66	5.66	0.00	7.55	**75.47**	1.89		
Vietnam	2.22	4.44	4.44	2.22	13.33	2.22	8.89	20.00	6.67	**35.56**		
Average											***47.30***	
STDV												***22.29***

Table 6.22: Results for SD-MF-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**15.69**	5.88	11.76	7.84	9.80	5.88	***35.29***	1.96	3.92	1.96		
English	2.20	**52.75**	2.20	2.20	5.49	6.59	17.58	1.10	7.69	2.20		
Japanese	2.22	4.44	**15.56**	2.22	8.89	***20.00***	***24.44***	2.22	8.89	11.11		
French	3.85	0.00	3.85	**63.46**	11.54	5.77	9.62	1.92	0.00	0.00		
Farsi	2.22	4.44	4.44	0.00	**66.67**	4.44	11.11	2.22	2.22	2.22		
Korean	0.00	0.00	9.76	0.00	14.63	**36.59**	24.39	0.00	7.32	7.32		
Mandarin	6.00	4.00	2.00	8.00	20.00	12.00	**40.00**	0.00	2.00	6.00		
Spanish	7.41	5.56	11.11	9.26	22.22	5.56	7.41	**24.07**	7.41	0.00		
Tamil	3.77	0.00	0.00	0.00	15.09	7.55	7.55	5.66	**56.60**	3.77		
Vietnam	2.22	0.00	6.67	2.22	15.56	0.00	***28.89***	2.22	15.56	**26.67**		
Average											***39.81***	
STDV												***19.24***

Table 6.23: Results for PLP-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

Confusion Matrix (%)	GE	EN	JA	FR	FA	KO	MA	SP	TA	VI	AVG	STDV
German	**17.65**	9.80	5.88	11.76	7.84	3.92	***19.61***	11.76	5.88	5.88		
English	6.59	**62.64**	2.20	1.10	1.10	3.30	12.09	2.20	6.59	2.20		
Japanese	2.22	6.67	**17.78**	6.67	2.22	15.56	***31.11***	8.89	4.44	4.44		
French	5.77	0.00	1.92	**55.77**	7.69	13.46	7.69	5.77	0.00	1.92		
Farsi	2.22	6.67	2.22	0.00	**55.56**	4.44	8.89	8.89	6.67	4.44		
Korean	4.88	4.88	2.44	0.00	17.07	**31.71**	17.07	9.76	7.32	4.88		
Mandarin	8.00	4.00	4.00	4.00	14.00	22.00	**40.00**	0.00	2.00	2.00		
Spanish	3.70	3.70	1.85	7.41	18.52	11.11	11.11	**37.04**	5.56	0.00		
Tamil	3.77	1.89	0.00	0.00	11.32	5.66	9.43	0.00	**58.49**	9.43		
Vietnam	4.44	6.67	2.22	2.22	8.89	0.00	24.44	4.44	17.78	**28.89**		
Average											***40.55***	
STDV												***16.79***

Table 6.24: Results for SD-PLP-CC Features - 10 Language Task.
Rows correspond to the actual class of the data files, columns to the assigned class for each file.

6.5 Repeatability/Consistency of Results

Sets of 5 sequential test runs of Feature Extraction, GMM Training, and Testing, were conducted for each of the feature vector types, as well as different parameter settings. Table 6.25 shows the results without Pre-Emphasis, Cepstral Speech Enhancement, or Cepstral Mean Subtraction. Table 6.26 shows the results with Pre-Emphasis, Cepstral Speech Enhancement, and Cepstral Mean Subtraction. All tests within each batch of 5 runs used the same set of software parameters in order to determine a general approximation as to how much variation in the results exists.

As evidenced by the tabulated results, the algorithm is able to reliably deliver consistent results to within a few percentage points accuracy. The tabulated percentages also reflect the empirical findings of this thesis that Shifted Delta Cepstral Coefficients generally can outperform regular cepstral coefficients. Also the tabulated results show that in our experiments, Shifted Delta Linear Predictive Cepstral Coefficients seem to perform the best overall. Furthermore, the tabulated results also serve to indicate that the inclusion of Pre-Emphasis, Cepstral Speech Enhancement, and Cepstral Mean Subtraction have a positive impact on the accuracy of the algorithm.

						Average	STDV
LP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	47.45	46.79	50.81	46.51	45.20	**47.35**	**2.10**
STDV Diag	15.45	17.85	12.17	17.68	19.41	**16.51**	**2.81**
SD-LP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	65.49	73.67	69.83	67.82	68.20	**69.00**	**3.04**
STDV Diag	16.01	10.48	16.73	12.75	11.71	**13.54**	**2.72**
MF-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	52.50	49.89	50.17	50.17	50.38	**50.62**	**1.06**
STDV Diag	22.24	23.68	23.79	23.79	20.43	**22.79**	**1.47**
SD-MF-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	63.26	63.55	65.10	64.90	67.24	**64.81**	**1.58**
STDV Diag	19.47	16.36	20.71	20.40	17.14	**18.82**	**1.96**
PLP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	47.64	50.40	48.82	47.53	46.69	**48.22**	**1.44**
STDV Diag	2.77	7.25	3.40	3.89	5.44	**4.55**	**1.80**
SD-PLP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	61.20	65.48	61.40	64.85	66.53	**63.89**	**2.44**
STDV Diag	7.14	10.57	7.49	4.41	6.29	**7.18**	**2.24**

Table 6.25: Amount of variation in results over five separate complete runs.
Cepstral Mean Subtraction (CMS), Cepstral Speech Enhancement (CSE), & Pre-Emphasis (PE) Filtering Disabled

						Average	STDV
LP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	59.11	59.85	59.11	58.00	59.31	**59.08**	**0.67**
STDV Diag	15.75	15.39	15.75	15.57	12.45	**14.98**	**1.42**
SD-LP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	71.05	69.74	72.98	71.13	70.77	**71.13**	**1.17**
STDV Diag	14.72	16.19	15.18	16.68	14.44	**15.44**	**0.96**
MF-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	53.64	55.11	53.36	54.13	55.03	**54.25**	**0.79**
STDV Diag	16.54	17.64	17.07	12.77	15.56	**15.92**	**1.92**
SD-MF-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	68.23	66.10	66.38	71.02	66.18	**67.58**	**2.11**
STDV Diag	20.04	21.24	19.78	18.04	23.31	**20.48**	**1.95**
PLP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	57.61	57.27	56.26	58.09	56.61	**57.17**	**0.74**
STDV Diag	18.30	14.31	11.53	15.90	15.04	**15.02**	**2.46**
SD-PLP-CC	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5		
Average Diag	67.16	65.49	66.23	64.82	64.89	**65.72**	**0.99**
STDV Diag	15.20	14.47	15.68	20.13	21.99	**17.49**	**3.35**

Table 6.26: Amount of variation in results over five separate complete runs.
Cepstral Mean Subtraction (CMS), Cepstral Speech Enhancement (CSE), & Pre-Emphasis (PE) Filtering Enabled

6.6 Effects of Amount of Training Data and Number of Mixtures on LID results

An experiment was also run using SD-LP-CC feature vectors to verify that the accuracy of the system is somewhat dependent on the amount of training data supplied for the Gaussian Mixture Models, as is predicted by GMM theory and discussed in section 4.2.

The plots generated in this set of experiments show a general tendency for accuracy to increase as training data increases, as is expected. Minor deviations from the increasing trend can be also attributed to the stochastic nature of the feature vector selection and GMM training.

As can be seen by the plots, the maximum average accuracy obtained was close to 80%.

This experiment was repeated for 16, 32, 64, 128 and 256 Mixture components, and as can be expected from theory, it can be clearly seen that higher mixture orders have a higher tendency for erroneous outlying data points at low amounts of training data. In essence, the cutoff point for the amount of training data that must be used in order to assure accurate results increases as the number of mixtures used increases.

The two major outlying data points in figure 6.4 at 400(s) and 750(s), and similar data points in the other plots, can be attributed to the stochastic nature of the NETLAB GMM initialization and training procedures, and the randomized feature vector selection, along with low amounts of training data being present. Once the amount of training data reaches significant levels, the erroneous major outliers are eliminated. Examples of this can be seen in all of the plots with 64 mixtures or more, and is not evident in the plots for 16 and 32 mixtures due to their low mixture level.

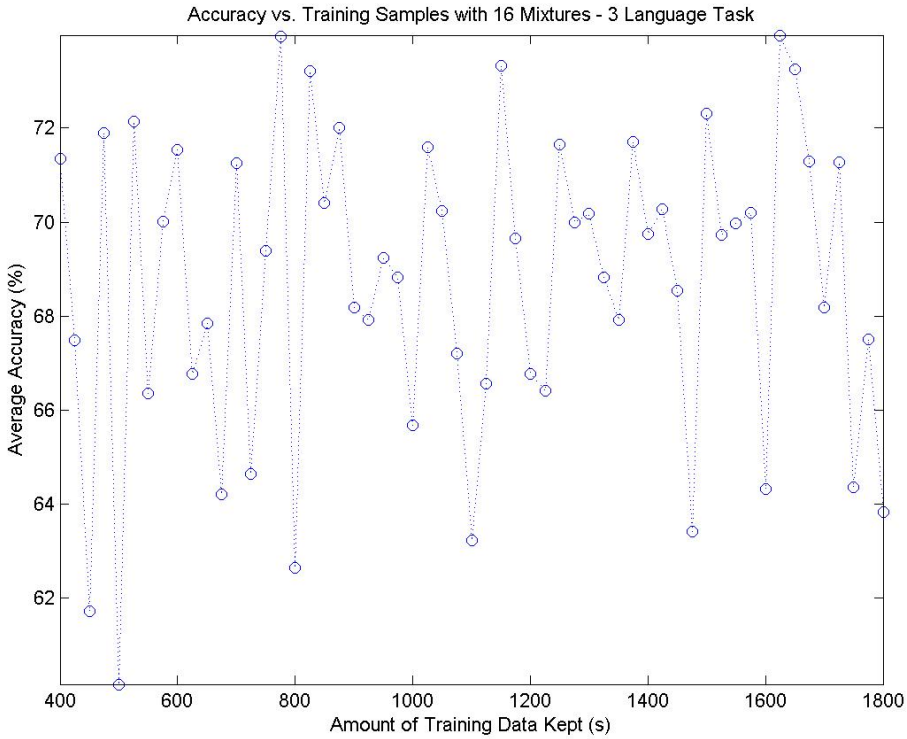

Figure 6.1: Plot of Training Data vs. Average Accuracy along Confusion Matrix Diagonal for SD-LP-CC feature vectors, 16 mixture components.

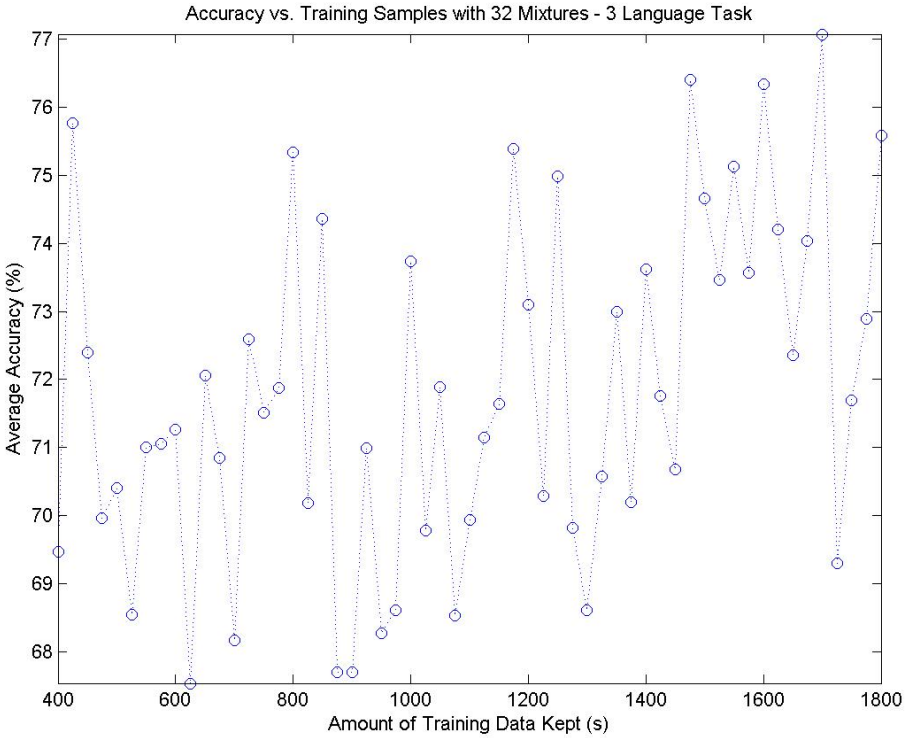

Figure 6.2: Plot of Training Data vs. Average Accuracy along Confusion Matrix Diagonal for SD-LP-CC feature vectors, 32 mixture components.

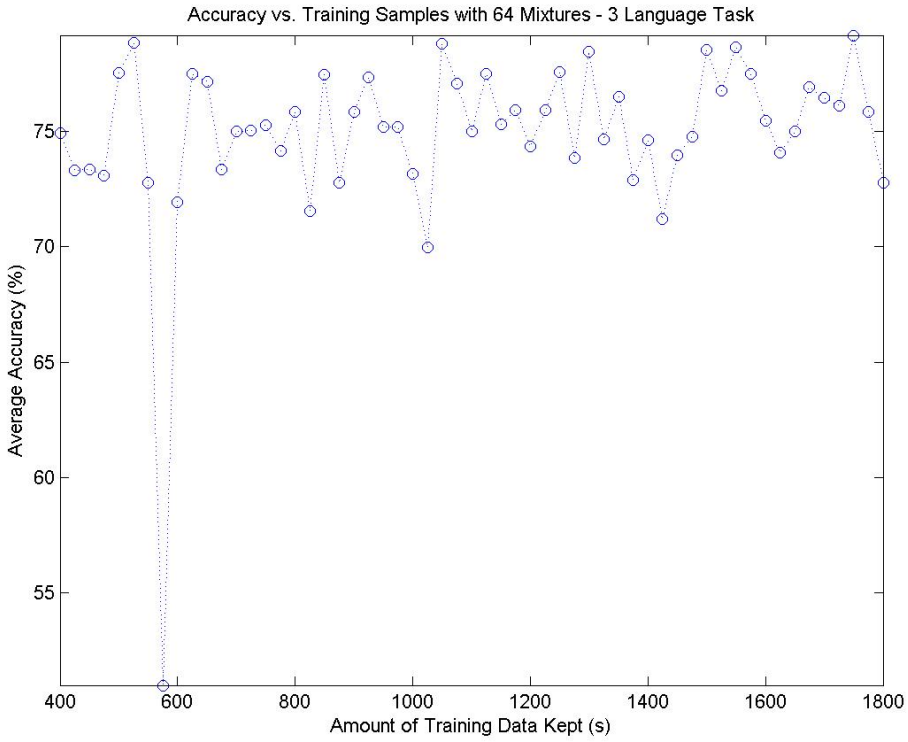

Figure 6.3: Plot of Training Data vs. Average Accuracy along Confusion Matrix Diagonal for SD-LP-CC feature vectors, 64 mixture components.

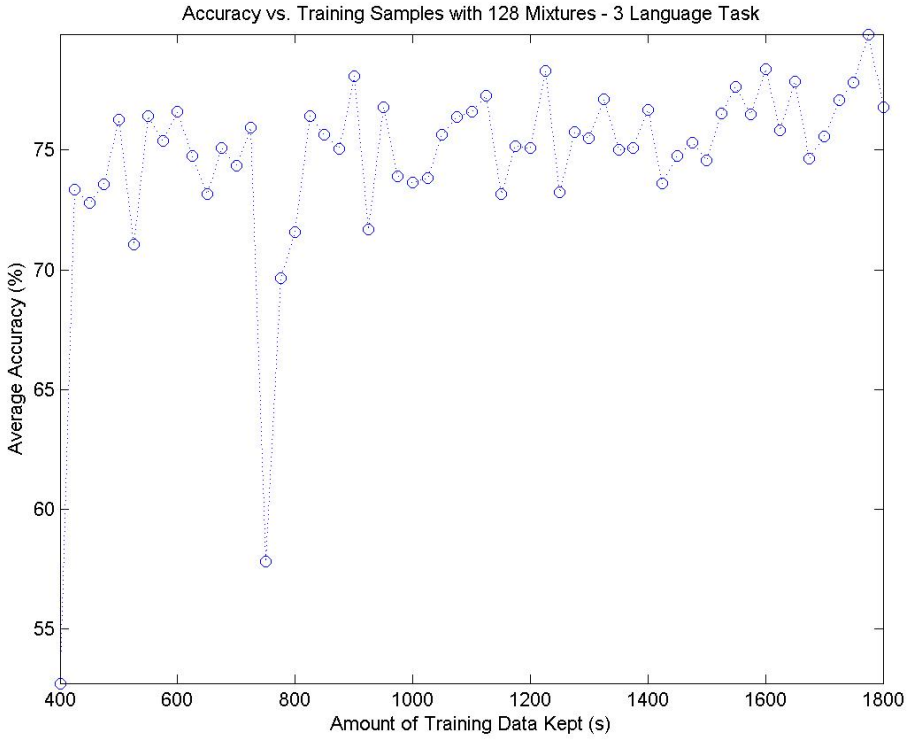

Figure 6.4: Plot of Training Data vs. Average Accuracy along Confusion Matrix Diagonal for SD-LP-CC feature vectors, 128 mixture components.

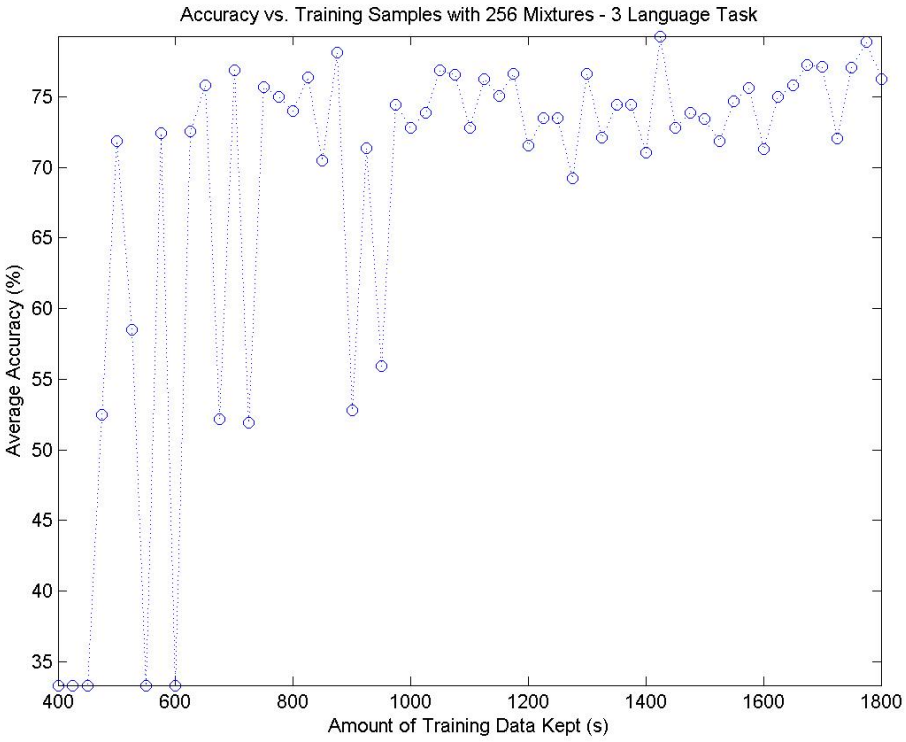

Figure 6.5: Plot of Training Data vs. Average Accuracy along Confusion Matrix Diagonal for SD-LP-CC feature vectors, 256 mixture components.

Chapter 7

Discussion and Future Work

7.1 Conclusion

The Shifted Delta Cepstra is a way of capturing pseudo-prosodic information from a speech signal and can be seen to improve language identification performance over standard cepstral coefficients in our experiments. In particular, when used in conjunction with Cepstral Mean Subtraction (CMS), Pre-Emphasis Filtering (PE), and Cepstral Speech Enhancement (CSE) the results show even greater improvement.

Based on the results obtained in this thesis, we can conclude that for this type of LID system there is a significant dependence on the method of computing cepstral features, and that SD-LP-CC feature vectors can outperform the other 5 feature vector types examined. The developed algorithm was able to achieve an averaged accuracy for SD-LP-CC feature vectors at 71.13% (see table 6.26), with the highest accuracy recorded approaching 80.00% when higher mixture orders and amounts of training data were used (see section 6.6). This does not necessarily mean that Linear Predictive Shifted Delta Cepstral coefficients are inherently always better for language processing tasks, but it does illustrate a specific example of Linear Predictive Shifted Delta Cepstra out-performing the other feature vectors considered with the given set of parameters.

7.1.1 Comparison of Results with Previous Works in Language Identification

While the results presented here do not compare with the more established PRLM methods, which have been shown to reach accuracies above 90%[51], they are a step in the right direction for creating easy to use alternatives to the phonemic modeling process and the requirement of using phonemically labeled data sets. Our result of 71.13% average accuracy shows a marked improvement over earlier attempts at performing language identification without significant *a priori* knowledge. Zissman was able to achieve only 65% on a 3 Language GMM task[51], while Pellegrino and Andre-Obrecht[21] report an accuracy of 68% on the same task as Zissman. Our results are in agreement with the earlier work on the use of Shifted Delta Cepstral features, where accuracies were reported between 70%-75%[2, 14, 20], and examines the effect of different types of cepstral derivations on their results.

Perhaps the most recent and similar previous work in the Language Identification literature, and therefore the most directly comparable, was presented by Wang and Qu in 2003[35]. They present results for a Gaussian Mixture Bigram Model in conjunction with a Universal Background Bigram Model on a 3 Language (English, Chinese and French) task from the OGI database. Their results show the GMBM-UBBM algorithm achieving 70.128% accuracy for 128 Mixture components. In comparison, our algorithm produced a comparable average accuracy while utilizing half of the number of mixture components, a lower amount of training data, and without using the extra Bigram or Universal Background Modeling.

In 2001, Wong and Sridharan[3] used a GMM with adapted Universal Background Model architecture to compare types of Linear Predictive and Mel Frequency derived feature vectors for language identification. Their general conclusion was that Linear Prediction derived feature vectors outperformed their Mel Frequency counterparts, and is in agreement with the data presented here. Wong and Sridharan reported accuracies ranging between 43%-60% on a 10 language task based on the OGI database. The authors also state that, for each feature vector type, they attempted to find the optimal parameter settings. Whereas in the experiments presented here, the parameter settings are kept as consistent as possible across all feature types.

Multiple papers on the topic of Language Identification (LID) that do not utilize the PRLM approach used either pair-wise evaluation tests, or similar evaluation schemes that relied on the system choosing between two choices at any given time. In a pure binary choice system, there is an inherent 50% chance of guessing accurately. Whereas in our trials, the tertiary nature of these experiments

makes the guess percentage 33.33%. Such discrepancies in architecture and methodology make direct comparisons difficult. Examples of some of the recent papers that rely on such binary evaluation schemes, or variations thereof, include:

- The work presented by Dan, Bingxi and Xin in 2002[47], a vector quantization approach was used to try to identify English and Mandarin Chinese, again drawing their samples from the OGI corpus. In their 2 language task results, the authors report accuracies of 61.54% and 66.67% for Linear Predictive derived coefficients.
- The use of predictive error histogram vectors for LID by Gu and Shibata[23] in 2003, who present accuracies of 60.8% for different speakers when trying to discern between English and Japanese speech.
- In 2003, Grieco and Pomales[8] presented a technique for using short duration speech samples and a sub-sound multi-feature transition matrix to classify languages. The present accuracies of 35% for a 12-language task and 71% for a binary decision task.
- In 2004 Herry, Gas, Sedogobo, and Zarader[27] presented an algorithm based on Neural networks for spoken language detection using the OGI Database. They report a global average score of 77.47% on pair-wise detection tasks between 10 languages.

7.2 Future Work

Ideas for future work and enhancements include:

- Performing Hill Climbing, or another similar procedure to determine the optimal parameter settings for all of the discussed features, and if certain feature types perform better when using different parameterization trade-offs.
- Examine methods for improving the algorithm across many different languages and not just the 3-language task studied here.
- Add Gender specific GMM capability for increased accuracy by explicitly modeling the statistical differences between male and female speakers in a given language.
- Examine the performance benefits of using a UBM-GMM with KL-Divergence distance metric.
- CMS Algorithm Improvement.

- CSE Algorithm Improvement.

Bibliography

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Pres, Great Clarendon Street, Oxford OX2 6DP, 1995.

[2] D.A. Reynolds-M.A. Kohler R.J. Greene J.R. Deller Jr. E. Singer, P.A. Torres-Carrasquillo. Approaches to language identification using gaussian mixture models andshifted delta cepstral features. *Proc. International Conference on Spoken Language Processing in Denver,CO, ISCA*, pages 33–36,82–92, September 2002.

[3] Sridha Sridharan Eddie Wong. Comparison of linear prediction cepstrum coefficients and mel-frequency cepstrum coefficients for language identification. *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong*, May 2001.

[4] Daniel P. W. Ellis. Plp and rasta and mfcc, and inversion in matlab. http://www.ee.columbia.edu/ dpwe/resources/matlab/rastamat/, 2005.

[5] R.E. Wohlford F.J. Goodman, A.F. Martin. Improved automatic language identification in noisy speech. *Acoustics, Speech, and Signal Processing, ICASSP*, 1989.

[6] J.T. Foil. Language identification using noisy speech. *ICASSP*, 1986.

[7] Frederick Williams Fred D Minifie, Thomas J. Hixon. *Normal Aspecks of Speech, Hearing, and Language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

[8] E.O. Grieco, J.J.; Pomales. Short segment automatic language identification using a multifeature-transition matrix approach. *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, Vol. 3:III–730–III–733, 25-28 May 2003.

[9] Tom Henderson. *The Physics Classroom*. GlenBrook South High School, Glenbrook, Illinois, http://www.glenbrook.k12.il.us/GBSSCI/PHYS/Class/sound/u11l4d.html, 1996-2004.

[10] H. Hermansky. Perceptual linear predictive (plp) analysis of speech. *J. Acoust. Soc. Am.*, vol. 87, no. 4:1738–1752, Apr 1990.

[11] A. Gray Jr. J. Markel. Fixed point truncation arithmatic implementation of a linear prediction autocorrelation vocoder. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, Vol. 22, Issue 4:273–262, Aug 1974.

[12] A.K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[13] A.H. Gray Jr. J.D. Markel. *Linear Prediction of Speech*. Springer-Verlag, Berlin, 1976.

[14] M. Kohler, M.A.; Kennedy. Language identification using shifted delta cepstra. *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposiumon*, 3:69–72, 4-7 August 2002.

[15] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, Englewood Cliffs, NJ, http://www-ccs.ucsd.edu/matlab/toolbox/signal/levinson.html, 1987.

[16] J. Makhoul. Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975.

[17] Frederic Bimbot Guillaume Gravier Mathieu Ben, Michael Betser. Speaker diarization using bottom-up clustering based on a parameter-derived distance between adapted gmm's. *In INTERSPEECH-2004*, pages 2329–2332, 2004.

[18] Ian T. Nabney. *NETLAB: Algorithms for pattern Recognition*. Advances in Pattern Recognition. Springer, 2002.

[19] A.V. Oppenheim and R.W. Schafer. *Discrete Time Signal Processing*. Prentice Hall, 1989.

[20] J.R. Deller Jr. P.A. Torres-Carrasquillo, D.A. Reynolds. Language identification using gaussian mixture model tokenization. *Proc. International Conference on Acoustics, Speech, and Signal Processingin Orlando, FL, IEEE*, pages 757–760, 13-17 May 2002.

[21] R. Pellegrino, F.; Andre-Obrecht. An unsupervised approach to language identification. *Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings., 1999 IEEE International Conference on*, Vol 2:833–836, 15-19 Mar 1999.

[22] J. Picone. Signal modeling techniques in speech recognition. *in Proc. IEEE*, vol. 81:1215–1247, Sept 1993.

[23] T. Qian-Rong Gu; Shibata. Speaker and text independent language identification using predictive error histogram vectors. *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, Vol 1:I–36–9, 6-10 April 2003.

[24] Steven L. Eddins Rafael C. Gonzalez, Richard E. Woods. *Digital Image Processing Using Matlab*. Pearson Education, Inc., 2004.

[25] J. Volkmann S. S. Stevens. The relation of pitch to frequency: A revised scale. *American Journal of Psychology*, Vol. 53, No. 3:329–353, 1940.

[26] R.W. Schafer and L.W. Rabiner. *Digital representations of speech signals*. Morgan Kaufmann Publishers Inc., 1990.

[27] Celestin Sedogbo Jean-Luc Zarader Sebastian Herry, Bruno Gas. Language detection by neural discrimination. *Interspeech 2004 - ICSLP 8th International Conference on Spoken Language Processing ICC Jeju, Jeju Island, Korea*, 4-8 Oct. 2004.

[28] Torres-Carrasquillo P. A. Gleason-T. P. Campbell W. M.and Reynolds D.A. Singer, E. Acoustic, phonetic, and discriminative approaches to automatic languagerecognition. *Proc. Eurospeech in Geneva, Switzerland, ISCA*, pages 1345–1348, 1-4 September 2003.

[29] M.M. Sondhi. New methods of pitch extraction. *IEEE Trans. Audio and Electroacoustics*, Vol. AU-16 No. 2:262–266, June 1968.

[30] E.B. Neumann S.S. Stevens, J. Volkmann. A scale for the measurement of the psychological magnitude of pitch. *Journal of the Acoustical Society of America*, Vol. 8, No. 3:185–190, 1937.

[31] F.G. Stremler. *Introduction to Communication Systems - Third Edition*. Addison-Wesley: USA, 1990.

[32] Hema A. Murthy T. Nagarajan. A pairwise multiple codebook approach to language identification. *Workshop on Spoken Language Processing. An ISCA Supported Event Mumbai, India.*, January 9-11 2003.

[33] Gleason T. P. Torres-Carrasquillo, P. A. and D. A. Reynolds. Dialect identification using gaussian mixture models. *Proc. Odyssey: The Speaker and Language Recognition Workshop in Toledo,Spain, ISCA*, pages 297–300, 31 May - 3 June 2004.

[34] L.; Nelson D. Umesh, S.; Cohen. Frequency warping and the mel scale. *Signal Processing Letters, IEEE*, vol.9,no.3:104–107, 2002.

[35] Dan Qu; Bingxi Wang. Automatic language identification based on gmbm-ubbm. *Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003 International Conference on*, pages 722–727, 26-29 Oct 2003.

[36] J.L. Mitchell W.B. Pennebaker. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, NY, 1993.

[37] Eric W. Weisstein. Convolution. *From MathWorld–A Wolfram Web Resource.*, 2005.

[38] Eric W. Weisstein. Fast fourier transform. *From MathWorld–A Wolfram Web Resource*, 2005.

[39] Eric W. Weisstein. Fourier transform. *From MathWorld–A Wolfram Web Resource.*, 2005.

[40] Eric W. Weisstein. Wiener-khinchin theorem. *From MathWorld–A Wolfram Web Resource*, 2005.

[41] Wikipedia. Discrete cosine transform — wikipedia, the free encyclopedia, 2006. [Online; accessed 10-August-2006].

[42] Wikipedia. Discrete fourier transform — wikipedia, the free encyclopedia, 2006. [Online; accessed 10-August-2006].

[43] Wikipedia. Obstruent — wikipedia, the free encyclopedia, 2006. [Online; accessed 8-August-2006].

[44] Wikipedia. Short-time fourier transform — wikipedia, the free encyclopedia, 2006. [Online; accessed 8-August-2006].

[45] Wikipedia. Sonorant — wikipedia, the free encyclopedia, 2006. [Online; accessed 8-August-2006].

[46] P.A. Torres-Carrasquillo-D.A. Reynolds W.M. Campbell, E. Singer. Language recognition with support vector machines. *Proc. Odyssey: The speaker and Language Recognition Workshop in Toledo Spain,ISCA*, pages 41–44, 31 May - 3 June 2004.

[47] Qu Dan; Wang Bingxi; Wei Xin. Language identification using vector quantization. *Signal Processing, 2002 6th International Conference on*, 1:492–495, 26-30 Aug 2002.

[48] E. Barnard Y. K. Muthusamy and R. A. Cole. Reviewing automatic language identification. *IEEE Signal Processing Magazine*, vol. 11, no. 4:33–41, 1994.

[49] R. A. Cole Y. K. Muthusamy and B. T. Oshika. The ogi multi-language telephone speech corpus. *Proceedings of the 1992 International Conference on Spoken Language Processing (ICSLP 92), Alberta*, October 1992.

[50] M. Zissman. Automatic language identification using gaussian mixture and hidden markov models,. *ICASSP-93*, 1993.

[51] M.A. Zissman. Comparison of four approaches to automatic language identification. *EEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 4 no. 1:31–44, Jan 1996.

Appendix A

Original Software For Language Identification

A.1 Training

```
function [GMMLangs] = JL_LID_Train(TrainDirs,LANGUAGES,...
    MAXFS,NORM_FEATS,func,funcArgs,SecondsOfSpeech,NumMixtures,ItrEM)
% Returns a cell array of Gaussian Mixture Models for each training set.
% <TrainDirs>   Cell array of directory strings.  For each directory, a GMM
%               will be trained using all of the .WAV files in that
%               directory.
% <LANGUAGES>   Cell array of descriptive strings that label each directory
% <MAXFS>       Maximum Sampling Frequency
% <NORM_FEATS>  [0|1] Whiten the feature vectors before training
% <func>        Function pointer to the feature vector extraction function.
% <funcArgs>    Arguments for the feature vector extraction function.
% <SecondsOfSpeech>    Amount of speech (s) to use for training. A value of
%               -1 uses all available frames.  Otherwise, if the
%               cumulative amount of frames returned by the feature
%               extraction function multiplied by the seconds represented
%               by each frame is greater than SecondsOfSpeech, frames of
%               feature vectors are randomly selected so that the number of
%               retained frames multiplied by the seconds represented
%               by each frame is aproximately equal to SecondsOfSpeech.
%               The retained frames are then used to train the GMM.
% <NumMixtures> Number of mixtures to be used for each GMM.
% <ItrEM>       Number of Iterations of EM algorithm to use.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com


%NOTE:  Need to change over to string-value pairs as a way of parameter
%passing...

FID = 1;
LangFeats = [];
%[SUCCESS,MESSAGE,MESSAGEID] = mkdir('MODELS');
```

```
if (length(LANGUAGES) ~= length(TrainDirs))
    error('length(LANGUAGES) must be == length(TrainDirs)')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get The features for each language training set...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for l = 1:numel(TrainDirs)
    d = TrainDirs{l};
    FEATS = [];

    %Get all the feature vectors for this language
    [FEATS, CT, secPFrame] = JL_BACKEND(d,...
        'func',func,'funcArgs',funcArgs,'MAXFS',MAXFS);

    if ((CT > SecondsOfSpeech ) && (SecondsOfSpeech ~= -1))
        FEATS = shuffle(FEATS,2);
        FEATS = FEATS(:,1:round(SecondsOfSpeech/secPFrame));
        CT = size(FEATS,2)*secPFrame;
    end

    fprintf(FID,['Cumulative time for ',LANGUAGES{l},' %.0f(s) \n'], CT);

    if NORM_FEATS
        %Make Zero Mean...
        mf = mean(FEATS')';
        FEATS = FEATS - mf(:,ones(1,size(FEATS,2)));

        %Divide Through by STD
        sf = std(FEATS')';
        sf(find(sf==0))=1;
        FEATS = FEATS ./ sf(:,ones(1,size(FEATS,2)));
    end

    LangFeats{end+1} = FEATS;
end
fprintf(FID,'\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Make and Store Each one of the Models
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
GMMLangs = [];
for i = 1:length(TrainDirs)
    GMMTemp = JL_MAKE_GMM(LangFeats{i},NumMixtures, ItrEM);
    GMMLangs{i} = GMMTemp;
    %save(['MODELS/GMM_LANG_',LANGUAGES{i}],'GMMTemp');
end
```

A.1.1 Back-end Functionality

```
function [FEATS,t,secPFrame] = JL_BACKEND(d, varargin)
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com


fn = dir(d);

MAXFS = 8000;
FEATS = [];
t =0;
SecondsOfSpeech = -1; %Unlimited
```

```
func = @JL_GET_FEATS;
funcArgs = [];

if nargin > 2
    args = varargin;
    if iscell(args{1})
        args = args{1};
    end
    nargs = length(args);

    %ProbFunction, PFunc_Args, TransMatV, TransMatH, iter, UnObsIdx)
    for i=1:2:nargs
        switch args{i},
            case 'func', func = args{i+1};
            case 'funcArgs', funcArgs = args{i+1};
            case 'MAXFS', MAXFS = args{i+1};
            case 'SecondsOfSpeech', SecondsOfSpeech = args{i+1};
            otherwise,
                error(['invalid argument name ' args{i}]);
        end
    end
end

for i = 1:numel(fn) %for each file in the directory
    fname = fn(i).name;
    if ((length(fname) > 3) && (strcmpi(fname(end-3:end), '.wav')))
        %is a .wav file so we willl process it...
        [mix,fs] = wavread([d,'/',fname]);

        %If necessary resample the .wav file so that it is at the
        %appropriate sampling frequency...
        if (fs > MAXFS)
            mix = resample(mix, MAXFS, fs);
            fs = MAXFS;
        end

        [feat, sec, secPFrame] = func(mix,fs,funcArgs);
        FEATS = [FEATS feat];
        t = t+sec; %Total Seconds of speech used for training
    end
end

%Should we limit the number of frames returned.  If so, randomly select
%which frames to keep by shuffling and returning only the first x number of
%frames.  x = round(SecondsOfSpeech/secPFrame)
if ((t > SecondsOfSpeech ) && (SecondsOfSpeech ~= -1))
    FEATS = shuffle(FEATS,2);
    FEATS = FEATS(:,1:round(SecondsOfSpeech/secPFrame));
    t = size(FEATS,2)*secPFrame;
end
```

A.1.2 Making the Gaussian Mixture Models Using NETLAB

```
function [mix, options, errlog] = JL_MAKE_GMM(feats, ncentres,itr)
%Make a GMM to describe the distribution given by <feats> using <ncentres>
%mixture components, and only itr EM iterations...
%
%Adapted from NETLAB demo by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

[inputdim,M] = size(feats);
if nargin < 3
    itr = 25
```

```
end
%NetLab GMM

mix = gmm(inputdim, ncentres, 'diag');

options = foptions;
options(1) = 1; % Prints out error values.
%options(3) = .1; %min change in log-loklihood to proceed
options(14) = 500; % Max. number of iterations.

disp('Initializing GMM using gmminit');
mix = gmminit(mix, feats', options); %Initialize with k-means

options(1) = 1; % Prints out error values.
options(5) = 1; % Prevent Covar values from collapsing...
options(14) = itr; % Max. number of iterations.

disp('Running EM for mixture model');
[mix, options, errlog] = gmmem(mix, feats', options);

%x = testingdata;
%prob = gmmprob(mix, x)
```

A.2 Testing

```
function [confMat,nFiles,fileList] = JL_LID_Test(varargin)
%Test each of the .wav files in the directories entered against each of the
%stored GMM's for each LANGUAGE.  Arguments are string-value pairs.
% <TestDirs>    Cell array of directory strings.  For each directory, each
%               .wav file in the directory will be tested against each of
%               the GMM's.
% <Languages>   Cell array of descriptive strings that label each GMM.
% <FeatFunc>    Function pointer to the feature vector extraction function.
% <FeatFuncArgs>Arguments for the feature vector extraction function.
% <GMMs>        The Gaussian Mixture Models for each language.
%
%These additional arguments are optional and/or have default preset values:
% <MaxFS>       (8000) Maximum Sampling Frequency
% <NormFeats>   [0|(1)] Whiten the feature vectors before training
% <GMMDistMode> Operating mode of the GMM distance function.('PROB')
% <GMMDistFunc> Function pointer for the distance function used to evaluate
%               the feature vectors on each GMM.  (@gmmdist)
% <GMMDistFuncArgs> Additional Arguments for the GMM distance function.([])
% <GMM-UBM>      (optional) Universal Background Model to use for distance
%               evaluation.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

nFiles = [];
fileList = [];

TestDirs = [];
LANGUAGES = [];
func = [];
funcArgs = [];
GMMLangs = [];
UBM = [];

%Optional Args...
GMMDistMode = 'Prob';
MAXFS = 8000;
```

```
NORM_FEATS = 1;
GMMDistFunc = @gmmdist;
GMMDistFuncArgs = [];

if (length(varargin)==8)
    %Older parameter entry
    TestDirs = varargin{1};
    LANGUAGES = varargin{2};
    MAXFS = varargin{3};
    NORM_FEATS = varargin{4};
    func = varargin{5};
    funcArgs = varargin{6};
    GMMDistMode = varargin{7};
    GMMLangs = varargin{8}
    %{
    if (isempty(GMMLangs))
        %Load the GMM Language Models...
        for i = 1:length(LANGUAGES)
            load(['MODELS/GMM_LANG_',LANGUAGES{i}]);
            GMMLangs{i} = gmmTemp;
        end
    end
    %}
else
    args = varargin;
    if iscell(args{1})
        args = args{1};
    end
    nargs = length(args);

    %ProbFunction, PFunc_Args, TransMatV, TransMatH, iter, UnObsIdx)
    for i=1:2:nargs
        switch args{i},
            case 'TestDirs', TestDirs = args{i+1};
            case 'Languages', LANGUAGES = args{i+1};
            case 'MaxFS', MAXFS = args{i+1};
            case 'NormFeats', NORM_FEATS = args{i+1};
            case 'FeatFunc', func = args{i+1};
            case 'FeatFuncArgs', funcArgs = args{i+1};
            case 'GMMDistMode', GMMDistMode = args{i+1};
            case 'GMM-UBM', UBM = args{i+1};
            case 'GMMs', GMMLangs = args{i+1};
            case 'GMMDistFunc', GMMDistFunc = args{i+1};
            case 'GMMDistFuncArgs', GMMDistFuncArgs = args{i+1};
            otherwise
                error([args{i},' is not a valid option']);
        end
    end
end

if isempty(func)
    func = @JL_GET_FEATS;
    funcArgs = [];
end

if ( isempty(TestDirs) || isempty(LANGUAGES) || isempty(GMMLangs) )
    error('You did not supply enough args');
end
confMat = zeros(length(TestDirs),length(LANGUAGES));

for l = 1:length(TestDirs);
    d = TestDirs{l};
    fn = dir(d);
    FEATS = [];
    for i = 1:numel(fn) %for each file in the directory
        fname = fn(i).name;
        if ((length(fname) > 3) && (strcmpi(fname(end-3:end), '.wav')))
            %is a .wav file so we willl process it...
```

```
            [mix,fs] = wavread([d,'/',fname]);
            if (fs > MAXFS)
                mix = resample(mix, MAXFS, fs);
                fs = MAXFS;
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %Get the features for this file...
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            feat = func(mix,fs,funcArgs);
            if NORM_FEATS
                %Make Zero Mean...
                mf = mean(feat')';
                feat = feat - mf(:,ones(1,size(feat,2)));

                %Divide Through by STD
                sf = std(feat')';
                sf(find(sf==0))=1;
                feat = feat ./ sf(:,ones(1,size(feat,2)));
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %Score this file against the models for each category
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            c = zeros(1,length(LANGUAGES)); %confidence values....

            %If using a UBM add functionality for it here....
            %if ~strcmpi(GMMDistMode,'PROB')
            %    UBM = JL_MAKE_GMM(feat,CodeBookSize,itr);
            %else
            %    UBM = [];
            %end

            for i = 1:length(LANGUAGES)
                GMM1 = GMMLangs{i};
                c(i) = sum(GMMDistFunc(GMM1,feat',GMMDistMode,UBM,GMMDistFuncArgs));
            end

            %Find the model that has the best score, and save that as the
            %category for this file...
            [Y,ind] = max(c);
            confMat(l,ind) = confMat(l,ind)+1;

            %Also output a list of filenames and their detected language
            fileList{end+1,1} = fname;
            fileList{end,2} = LANGUAGES{ind};
        end
    end
    nFiles(l) = sum(confMat(l,:));
    %Turn confusion matrix into percentages (divide each row by its sum)
    confMat(l,:) = 100*confMat(l,:) / nFiles(l);
end
```

A.3 Default Feature Extraction using RASTA-MAT

```
function [feat,SecondsOfSpeech,SecondsPerFrame] = JL_GET_FEATS(mix,fs,varargin)
%  Gets the feature vectors for the signal MIX with sampling frequency FS
%  and parameters supplied by the set of string-value pairs given in
%  VARARGIN.
%
%Parameters: Brackets denote an optional parameter.  Parentheses denote
%           the default setting for that parameter.
%'print'            [(0)|1]  Display incremental output.
```

```
%'nfft'             (256) Size of FFT to use when calculating features
%'win'              (256) Size of Window to use when calculating features
%'ov'               (128) Amount of Overlap between calculation windows
%'minHz'            (300) Minimum frequency to allow.  300Hz is the
%                       standard cutoff frequency for telephone speech high
%                       pass filter.
%'numCoeff'         (12) Number of Cepstral Coefficients to use
%'numCoeffLP'       (12) Number of LP Coefficients to use when calculating
%                       LP-CC features.
%'PRE_EMPH'         [(0)|1] Use Pre-Emphasis Filtering to adjust for
%                       natural ~20dB/decade rolloff of the human voice.
%'ENHANCE'          [0|(1)] Use Cepstral Speech Enhancement algorithm as a
%                       pre-processor.
%'USE_CMS'          [0|(1)] Use Cepstral Mean Subtraction to try to
%                       mitigate channel effects.
%'USE_SDC'          [0|(1)] Use Shifted Delta Cepstral Coefficients
%'USE_DELTA'        [(0)|1] Use Delta Coefficients
%'USE_POWER_TERM'   [(0)|1] Use or omit the power (first) cepstral term
%'deltaDist'        ([]) The distance (in feature frames) to use when
%                       calculating delta coefficients.
%'SDC_Block_Spacing' ([]) The distance (in frames) to use between shifted
%                       delta blocks.
%'deltaDist_Sec'    (.1920) The distance (in seconds) to use when
%                       calculating delta coefficients.
%'SDC_Block_Spacing_Sec' (.048) The distance (in seconds) to use between
%                       shifted delta blocks.
%'SDC_Blocks'       (3) The Number of Shifted Delta Blocks to Use
%'LifterExp'        (0) The exponent to use when liftering (i.e. weighting)
%                       the cepstral coefficients
%'Mode'             ['LP-CC'|'PLP-CC'|'MF-CC'|'CUST'] Feature Calculation
%                       mode to use.  'LP-CC' - Linear Prediction derived
%                       Cepstral Coefficients.  'PLP-CC' - Perceptual
%                       Linear Prediction Cepstral Coefficients.  'MF-CC' -
%                       Mel Frequency Cepstral Coefficients.  'CUST' -
%                       Custom cepstral coefficient derivation function.
%                       Use of the 'CUST' option means one must also define
%                       'CustModeFunction' and 'CustModeFunctionArgs'.
%'CustModeFunction' ([]) A function handle to a user defined function for
%                       calculating the cepstral coefficients that obey the
%                       following prototype:
%                           CepCoeffs = FUNC(mix,fs,args)
%'CustModeFunctionArgs' ([]) Supplimental arguments for the custom mode
%                       function.
%'VTHRESH'          (.00056234) Threshold for determining if a
%                       frame is speech/non-speech data. .00056234 is
%                       equivalent to -65dB.
%'V_Max'            [(0)|1] Only use the locations of peaks in the
%                       sequential power signature of feature frames to
%                       extract speech frames.  Greatly reduced the number
%                       of feature frames extracted, but helps to ensure
%                       that only features frames corresponding to actual
%                       speech data are used.  Was implemented for
%                       debugging purposes only.  Not Recommended.
%'VOP'              [(0)|1] Try to use the locations of Vocal Onset Points
%                       to determine which frames are extracted from te
%                       audio signal. Again, was implemented as a debugging
%                       tool and is not recommended.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

lifterexp = 0;  %0 = No Liftering, 1 = Linear weighting ...
NEW = 0; %Use newer LP-CC computation code...
nfft = 256; win =  256; ov = win/2;
minHz = 300;
numCoeff = 12;
numCoeffLP = 12;
```

```
USE_PLP = 1;
USE_RASTA = 0;
USE_CMS = 1;
PRINT = 0;
USE_DELTA = 0;
PRE_EMPH = 0;
VTHRESH = .00056234;
USE_V_MAX = 0;
USE_V = 1;
USE_SDC = 0;
USE_POWER_TERM = 0;
mode = 'LP-CC';
ENHANCE = 1;
deltaDist = [];
ShiftedDeltaSpacing = [];
NumShiftedDeltas = 3;
CustModeFunction = [];
CustModeFunctionArgs = [];
deltaDistSec = .1920; %3 frames at win= 256 ov= 128 fs= 8000
ShiftedDeltaSpacingSec =.048;  %3 frames at win= 256 ov= 128 fs= 8000

%Added Jul 13th...
%Normalize the input...
%mix = mix / mean(mix(:));
%mix = mix / max(abs(mix(:)));

if (nargin > 2 && ~isempty(varargin))
    args = varargin;
    if iscell(args{1})
        args = args{1};
    end
    nargs = length(args);

    %ProbFunction, PFunc_Args, TransMatV, TransMatH, iter, UnObsIdx)
    for i=1:2:nargs
        switch args{i},
            case 'print', PRINT = args{i+1};
            case 'nfft', nfft = args{i+1};
            case 'win', win = args{i+1};
            case 'ov', ov = args{i+1};
            case 'numCoeff',  numCoeff = args{i+1};
            case 'numCoeffLP',  numCoeffLP = args{i+1};
            case 'USE_RASTA', USE_RASTA = args{i+1} ;
            case 'USE_CMS', USE_CMS = args{i+1} ;
            case 'PRE_EMPH', PRE_EMPH = args{i+1} ;
            case 'ENHANCE', ENHANCE = args{i+1} ;
            case 'USE_SDC', USE_SDC = args{i+1} ;
            case 'USE_DELTA', USE_DELTA = args{i+1};
            case 'deltaDist', deltaDist = args{i+1} ;
            case 'SDC_Block_Spacing',ShiftedDeltaSpacing = args{i+1};
            case 'SDC_Blocks',NumShiftedDeltas = args{i+1};
            case 'LifterExp',lifterexp = args{i+1};
            case 'UseNewer_LP-CC_Code', NEW = args{i+1};
            case 'deltaDist_Sec',
                deltaDist=[];
                deltaDistSec = args{i+1} ;
            case 'SDC_Block_Spacing_Sec',
                ShiftedDeltaSpacing=[];
                ShiftedDeltaSpacingSec = args{i+1};
            case 'Mode', mode = args{i+1}; %['LP-CC'|'PLP-CC'|'MF-CC'|'CUST']
            case 'CustModeFunction', CustModeFunction = args{i+1};
            case 'CustModeFunctionArgs', CustModeFunctionArgs = args{i+1};
            case 'minHz',  minHz = args{i+1};
            case 'Print',  PRINT = args{i+1};
            case 'VTHRESH', VTHRESH = args{i+1};
            case 'USE_POWER_TERM', USE_POWER_TERM = args{i+1};
            case 'V_Max', USE_V_MAX = args{i+1};
                if USE_V_MAX
```

```
                    USE_VOP = 0;
                    USE_V = 0;
                end
            case 'VOP', USE_VOP = args{i+1};
                if USE_VOP
                    USE_V = 0;
                    USE_V_MAX = 0;
                end
            otherwise,
                error(['invalid argument name ' args{i}]);
        end
    end
end

if isempty(deltaDist)
    deltaDist = max(1,round(deltaDistSec*fs/(win-ov)));
end
if isempty(ShiftedDeltaSpacing)
    ShiftedDeltaSpacing = max(1,round(ShiftedDeltaSpacingSec*fs/(win-ov)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Pre-processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Normalize the file...
mix = mix / max(abs(mix(:)));

if ENHANCE
    %Clean up the telephone speech...
    mix = cepFilt(mix,nfft,fs,win,ov,minHz,0);
end

if PRE_EMPH
    %Pre-Emphasis...
    A = [1 -.95];
    B = [1];
    mix = filter(A,B,mix);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NSeconds = length(mix)/fs;  %Length of the Full file (s)
SecondsPerFrame = win/fs;   %Seconds of spech data contained in each frame.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get the Features for each frame of the audio signal....
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if strcmpi(mode,'PLP-CC')
    %Use Ellis RASTA-MAT package to computer PLP-CC
    [cepCoeffs, spectra, pspectrum, lpcas] = ...
        rastaplp(mix, fs, USE_RASTA, numCoeff, win, ov, lifterexp);
elseif strcmpi(mode,'MF-CC')
    %Use Ellis RASTA-MAT package to computer MF-CC
    [cepCoeffs,aspectrum,pspectrum] = melfcc(mix, fs, 'wintime', win/fs,...
        'hoptime',(win-ov)/fs,'lifterexp',lifterexp,'numcep',numCoeff);
elseif strcmpi(mode,'LP-CC')
    %Turn into frames
    mixf = makeframes(mix,win,ov, 'hamming');

    %Get LP-CC coefficients...
    if NEW  %Uses the Ellis package to compute LP-CC's...
        %Get the Linear Predictive Coefficients
        [lpcas] = dolpc(mixf,numCoeffLP);

        %Use Recursion to find Cepstral Coefficients
        cepCoeffs = lpc2cep(lpcas',numCoeff);
```

```
        %Apply Liftering if needed
        cepCoeffs = lifter(cepCoeffs, lifterexp);  %NEW Consistant with others
    else  %older method, more direct method...
        %Get the Linear Predictive Coefficients
        [lpcas, lpcErrPow] = lpc(mixf,numCoeffLP);

        %Get the frequency spectra made from the LPC coefficients using the
        %signa processing toolbox functions...
        pspectrum = zeros(nfft/2+1,size(mixf,2));
        for j = 1:size(mixf,2)
            [B,A] = eqtflength(1,[0 lpcas(j,:)]);
            pspectrum(:,j) = freqz(B,A,nfft/2+1,fs);
        end

        %Normalize the spectrum
        pspectrum = pspectrum / max(abs(pspectrum(:)));

        %Convert the Fourier Spectra to N Cepstral Coeff's...
        cepCoeffs = real(ifft(log(abs(pspectrum)), nfft));

        %Get the Linearly Weighted (Liftered) Cepstral Coefficients's for
        %each frame, and also only retain the number of
        %coefficients that we want...
        mul = 1:numCoeff;
        if (lifterexp ~= 0)
            mul1 = mul.^lifterexp;
            cepCoeffs = cepCoeffs(mul,:).*mul1(ones(size(cepCoeffs,2),1),:)';
        else
            cepCoeffs = cepCoeffs(mul,:);
        end
    end
else
    if isa(CustModeFunction, 'function_handle')
        %Use a custom feature extraction method...
        cepCoeffs = CustModeFunction(mix,fs,CustModeFunctionArgs);
    else
        error('In order to use the Custom feature calculation mode, you must supply a function handle');
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Post-Processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~USE_POWER_TERM
    %Omit the power coefficient...
    cepCoeffs = cepCoeffs(2:end,:);
end

%Try to remove channel effects by doing Cepstral Mean Subtraction...
if USE_CMS
    %Might want to make this a little more advanced via sliding window,
    %etc...
    cepCoeffs = cepCoeffs - repmat(mean(cepCoeffs,2),1,size(cepCoeffs,2));
end

feat = [cepCoeffs];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Figure out which feature frames to remove (silence) and make delta cepstra
%if applicable...
if (USE_V_MAX || USE_V)
    %Do we want to use the Delta Cepstra...
    if (USE_DELTA || USE_SDC)
        %Get the delta-Cepstra
        DeltaCepCoeffs = deltas(cepCoeffs,deltaDist);

        %If we need to compute the Shifted-Delta Cepstra...
        if USE_SDC
```

```
            %P = shift between blocks
            %k = number of blocks
            %N = numcoeff
            %d = deltaDist
            P = ShiftedDeltaSpacing;
            k = NumShiftedDeltas;
            feat = [DeltaCepCoeffs];
            for i = 2:k
                shift = (k-1)*P;
                feat = [feat; circshift(DeltaCepCoeffs, [0 -shift])];
            end

            %Remove frames that have wrapped around...
            feat = feat(:,1:end-shift);
        else
            feat = [feat; DeltaCepCoeffs]; %add any other additional features here...
        end
    end

    %Locate using power, and get rid of silent frames.  Basing on the
    %derived power spectrum can introduce inconsistancies between feature
    %types, so instead we use the normalized original signal to make
    %speech/non-speech determination...
    %[V1, Vm1, t1] = voicingDetector(pspectrum,[],0,VTHRESH,0);
    [V, Vm, t] = voicingDetector(mix,fs,0,VTHRESH,0);

    FrameSecLabels = (1:size(feat,2))*(NSeconds/size(feat,2));

    if USE_V_MAX %Not Recommended
        t(find(Vm'==0)) = [];
    else
        %feat(:,find(V==0)) = [];
        t(find(V'==0)) = [];
    end
    %Locate which frames correspond to speech.
    npts = nearestpoint( t, FrameSecLabels);

    %Remove any duplicates
    npts = removeDuplicates(npts);

    %only keep those concatonated feature vectors that correspond to the
    %speech segments and disregard the other (silence) frames
    feat = feat(:,npts);

elseif USE_VOP
    %We use the location of Vocal onset points, and concatonate subsequent
    %frames of features around the VOP...
    spacing = 1;

    feat = [feat; circshift(feat,[0,1*spacing]);  circshift(feat,[0,-1*spacing]);...
        circshift(feat,[0,-2*spacing]);  circshift(feat,[0,-3*spacing])];

    %Do we want to use the Delta Cepstra...
    if USE_DELTA
        %Get the delta-Cepstra
        DeltaCepCoeffs = deltas(cepCoeffs,deltaDist);

        feat = [feat; DeltaCepCoeffs]; %add any other additional features here...
    end

    %Find Locations of Vocal Onset Points...
    [VOP_Times, VOP, dVOP] = VocalOnsetPoints(mix, fs, win, ov, numCoeffLP);
    FrameSecLabels = (1:size(feat,2))*(NSeconds/size(feat,2));

    %Locate which frames correspond to the Vocal Onset Points.
    npts = nearestpoint( VOP_Times, FrameSecLabels);
    npts = removeDuplicates(npts);
```

```
    %only keep those concatonated feature vectors that correspond to the
    %VOP's
    feat = feat(:,npts);
else
    error('Cannot determine which voicing Algorithm to use');
end

%How much speech has been kept?
SecondsOfSpeech = size(feat,2)*SecondsPerFrame;
if PRINT
    disp(['FEATS Size: ',num2str(SecondsOfSpeech),' (s)']);
end
```

A.4 Default GMM Distance Metric

```
function d = gmmdist(GMM1,feat,mode,GMM2,varargin);
%Default GMM distance calculation function. For each feature vector the
%function returns a distnace <d>.  Custom distance functions must follow
%the same parameter passing scheme.
%<GMM1>     Primary Gaussian Mixture Model used to evaluate each vector in
%           <feat>.
%<feat>     Set of feature vectors to evaluate.  [N by M] with N being the
%           number of feature vectors and M being each vector length.
%<mode>     [('PROB') | 'Sym-KL' | 'KL']
%           - 'PROB' - Uses the probability of each feature vector
%           falling on GMM1.  This is the most basic and straightforward
%           distance metric. GMM2 is ignored in this mode.
%           - 'KL' - Uses the Kullback-Liebler Divergence to calculate the
%           asymetric distance of the features between GMM1 and GMM2.
%           - 'Sym-KL' - Uses the Kullback-Liebler Divergence to calculate
%           the symetric distance of the features between GMM1 and GMM2.
%<GMM2>     Secondary Gaussian Mizture Model used for KL distance modes.
%<varargin> Is ignored in this default distance function.  Intended so that
%           custom distance functions can pass additional parameters if
%           needed.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com


if (nargin < 3 || isempty(mode))
    mode = 'PROB';
end

if (nargin < 4 || isempty(GMM2))
    %If the GMM for the feats is not provided, automatically return the
    %PROB.
    mode = 'PROB';
end

if strcmpi(mode,'Sym-KL')
    %Symmetric KL Divergance
    d = -.5*(KL(GMM1,GMM2)+KL(GMM2,GMM1));
elseif strcmpi(mode,'PROB')
    %Probability distance metric...
    d = log(gmmprob(GMM1,feat));
    %d = log(gmmprob(GMM1,feat)+eps);

elseif strcmpi(mode,'KL')
    %KL-Divergence
    d = -KL(GMM1,GMM2);
else
```

```
    error('Dist Mode Not Valid');
end
```

A.5 Example Script for Running an Experiment

```
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

clear;

tic;

func = @JL_GET_FEATS;  %Default function to use to generate features
funcArgs = [];

TrainDirs = [];
TestDirs = [];
VerDirs = [];

%Should we normalize the features...no real reason to change this, but
%the option is there
NORM_FEATS = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%These flags and varuiables can be changed to alter performance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CodeBookSize = 64;      %Number of mixtures to use in GMM
SecondsOfSpeech = 1800; %Cumulative amount of speech to use for training
mode = 'LP-CC';         %['LP-CC' | 'MF-CC' | 'PLP-CC' | UserDefined]
GMMDistMode = 'PROB';   %Distance Metric to use...
USE_CMS = 1;            %Use Cepstral Mean Subtraction for Channel Equalization
ENHANCE = 1;            %Use Enhancement Function (NOTE: make a func ptr later)
PRE_EMPH = 0;           %Use Pre-emphasis filter
VTHRESH = .00056234;    %Voicing Threshold = -65dB
V_MAX = 0;              %Use only the Max Locations in Voicing Detector
VOP = 0;                %Use the Vocal Onset Point
PRINT = 1;              %Display incremental output
USE_DELTA = 0;          %Use Delta Cepstra
USE_RASTA = 0;          %Use Rasta (only applicable when using PLP)
USE_SDC = 1;            %Use Shifted Delta Cepstra...
deltaDistSec = .1920;    %=12 frames at win= 256 ov= 128 fs= 8000
ShiftedDeltaSpacingSec=.048;  %=3 frames at win= 256 ov= 128 fs= 8000
NumShiftedDeltas = 3;   %Number of delta Blocks
numCoeff = 12;          %Number of Cepstral Coefficents to use
numCoeffLP = 12;        %Number of LP Coefficients to use(Valid with 'LP-CC' Mode)
itr = 10;               %Number of GMMEM iterations before breaking...

nfft = 256; win = 256; ov = 128;
minHz = 300; framerate = 100; MAXFS = 8000;

TrainHD = 'D:/Jons Files/Test Data/OGI_TEST_SETS/stb/Train/';
TestHD = 'D:/Jons Files/Test Data/OGI_TEST_SETS/stb/Test/';

LANGUAGES = [];
LANGUAGES{end+1} = 'GERMAN';
LANGUAGES{end+1} = 'ENGLISH';
LANGUAGES{end+1} = 'JAPANESE';
%LANGUAGES{end+1} = 'FRENCH';
%LANGUAGES{end+1} = 'FARSI';
%LANGUAGES{end+1} = 'KOREAN';
%LANGUAGES{end+1} = 'MANDARIN';
```

```
%LANGUAGES{end+1} = 'SPANISH';
%LANGUAGES{end+1} = 'TAMIL';
%LANGUAGES{end+1} = 'VIETNAM';
nLANGUAGES = length(LANGUAGES);

for i = 1:nLANGUAGES
    %Closed (training) set...
    TrainDirs{end+1} = [TrainHD,LANGUAGES{i}];

    %And the open set...
    TestDirs{end+1} = [TestHD,LANGUAGES{i}];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Running the algorithm...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isempty(funcArgs)
    try
        funcArgs =  {'nfft',nfft,'win',win,'ov',ov,'minHz',minHz,...
            'numCoeff',numCoeff,'numCoeffLP',numCoeffLP,'USE_RASTA',USE_RASTA,...
            'print',PRINT,'USE_DELTA',USE_DELTA,...
            'VTHRESH',VTHRESH,'Mode',mode, 'V_Max', V_MAX, ...
            'VOP', VOP,'USE_SDC',USE_SDC,'USE_CMS',USE_CMS...
            'ENHANCE',ENHANCE,'PRE_EMPH',PRE_EMPH,...
            'deltaDist_Sec',deltaDistSec,...
            'SDC_Block_Spacing_Sec',ShiftedDeltaSpacingSec,...
            'SDC_Blocks',NumShiftedDeltas...
            };
    catch
        error('You did not supply all necessary arguments');
    end
end

disp('running')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Train - Make arguments string value pairs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[GMMLangs] = JL_LID_Train(TrainDirs,LANGUAGES,...
    MAXFS,NORM_FEATS,func,funcArgs,SecondsOfSpeech,CodeBookSize,itr);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Test - Make Arguments string value pairs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[confMat,nFiles,fileList] = JL_LID_Test(TestDirs,LANGUAGES,...
    MAXFS,NORM_FEATS,func,funcArgs,GMMDistMode,GMMLangs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Print Results to Screen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(1,'\nConfusion Matrix Open Set: \n');
for i = 1:nLANGUAGES
    fprintf(1,'%s (%.0f Files):',LANGUAGES{i}(1:2),nFiles(i));
    for j = 1:size(confMat,2)
        fprintf(1,'\t%2.2f', confMat(i,j));
    end
    fprintf(1,'\n');
end
ad = mean(diag(confMat));
sd  = std(diag(confMat));

fprintf(1,'Avg Diag: ');
fprintf(1,'\t%2.2f\n', ad);
fprintf(1,'STDV Diag: ');
fprintf(1,'\t%2.2f\n', sd);

toc
```

A.6 Utilities and Other Functions/Sub-Functions

A.6.1 GMM Demo Function

```
function GMMDemo
%Demo showcasing Gaussian Mixture Models.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

%Make Sample Data
rad = 0:.009:2*pi;
mag = rand(size(rad))+5;

y = mag.*sin(rad);
x = mag.*cos(rad);
feat = [x;y];

subplot(2,2,1),plot(x,y,'b. '); axis square; axis([-7,7,-7,7]);
title('Raw Data Points');
drawnow;

%Now make GMM approximations to the data distribution using various numbers
%of mixtures...

ncentres = 4;
subplot(2,2,2),
[mix, options, errlog] = JL_MAKE_GMM(feat,ncentres);
% Plot the result
x = -7.0:0.2:7.0;
y = -7.0:0.2:7.0;
[X, Y] = meshgrid(x,y);
X = X(:);
Y = Y(:);
grid = [X Y];
Z = gmmprob(mix, grid);
Z = reshape(Z, length(x), length(y));
c = mesh(x, y, Z);
view(2);
axis square; axis tight;
hold on
title(['GMM N:',num2str(ncentres)]);
hold off
drawnow;

ncentres = 16;
subplot(2,2,3),
[mix, options, errlog] = JL_MAKE_GMM(feat,ncentres);
% Plot the result
x = -7.0:0.2:7.0;
y = -7.0:0.2:7.0;
[X, Y] = meshgrid(x,y);
X = X(:);
Y = Y(:);
grid = [X Y];
Z = gmmprob(mix, grid);
Z = reshape(Z, length(x), length(y));
c = mesh(x, y, Z);
view(2);
axis square; axis tight;
hold on
title(['GMM N:',num2str(ncentres)]);
hold off
drawnow;
```

```
ncentres = 32;
subplot(2,2,4),
[mix, options, errlog] = JL_MAKE_GMM(feat,ncentres);
% Plot the result
x = -7.0:0.2:7.0;
y = -7.0:0.2:7.0;
[X, Y] = meshgrid(x,y);
X = X(:);
Y = Y(:);
grid = [X Y];
Z = gmmprob(mix, grid);
Z = reshape(Z, length(x), length(y));
c = mesh(x, y, Z);
view(2);
axis square; axis tight;
hold on
title(['GMM N:',num2str(ncentres)]);
hold off
drawnow;
```

A.6.2 Cepstral Speech Enhancement

```
function [xCout, xHout, xFout, xout] = cepFilt(x,nfft,fs,win,ov, minHz, PRINT)
%Do Cepstral Speech Enhancement on mono audio signal <x>
%<nfft>     - Size FFFT to use.
%<fs>       - Sampling Frequency
%<win>      - window size
%<ov>       - Overlap
%<minHz>    - minimum frequency (LPF cutoff)
%<PRINT>    - Display...
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

%dh = 'D:\Jons Files\Thesis\JJL_MS_Thesis\Language_Samples\Hindi\hindi_spkr_4.wav';
%de = 'D:\Jons Files\Thesis\JJL_MS_Thesis\Language_Samples\English\eng_spkr_1.wav';

%[x,fs] = wavread(x);
%For Debugging...
if nargin < 1
    PRINT = 1;
    nfft = 256;
    win = 256;
    ov = 237;
    minHz = 300;
    de = 'D:\Jons Files\Test Data\OGI_TEST_SETS\Train\ENGLISH\EN003DOW.waV'
    %de = 'D:\Jons Files\Test Data\KalmanFilteringSpeechEnhancement\Orig.wav';
    %    de = 'D:\Jons Files\Thesis\JJL_MS_Thesis\Language_Samples\English\Full\eng_spkr_1.wav';
    [x,fs] = wavread(de);
    x = resample(x,8000,fs);
    %x = x(.37*8000:1.3*8000);
    fs = 8000;
else

    if nargin < 6
        minHz = 300;
    end

    if nargin < 7
        PRINT = 0;
    end
end
```

```
mixFH = .85 ;
CepThresh = .9;
GaussAlpha = 5 ;

x = x(:);
if (minHz > 0)
    [B,A] = butter(3,minHz/(fs/2),'high');
    x = filtfilt(B,A,x);
end
x = x / max(abs(x(:)));

%Get the unfiltered spectrogram
XE = makeframes(x, win, ov, 'hamming');
fe = fft(XE,nfft);
fe = fe / (max(fe(:))+eps);

b = fe(1:(nfft/2+1),:);
f = 1:fs/(2*(nfft/2+1)):fs/2;
t = 0:((length(x)/fs)/size(b,2)):(length(x)/fs);

%Compute the Cepstrum...
ce = real(ifft(20*log10(abs(fe)), nfft));

szce2 = size(ce,1);
T = -floor(szce2/2):floor(szce2/2)-1;
ind = round((1*fs)/length(x) * size(ce,2));
size(T);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Filter...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
frm = ce;
%Create a gaussian window to isolate the formants
w = gausswin(size(ce,1),GaussAlpha);
frmF = fftshift(fftshift(ce).*(w(:,ones(1,size(ce,2))) ));

%Subtract the isolated formants from the full Cepstrum to find pitch peaks
frmGm = frm - frmF;
%Find the maximums
[frmLmX, frmLmY] = localmax(frmGm);
%Keep only those peaks that are greater than the threshold
frmH = frmGm.*(frmGm.*frmLmY > CepThresh);

CepCombined = frmH+frmF; %Apporximate 'Clean' Cepstrum...

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Invert back into fourier spectrum...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Make minimum phase...
w = [1; 2*ones(nfft/2-1,1); ones(1 - rem(nfft,2),1); zeros(nfft/2-1,1)];
w = w(:,ones(1,size(CepCombined,2)));
%Compute FFT
frme = (fft(CepCombined.*w,nfft));
frmeF = (fft(frmF.*w,nfft));
frmeH = (fft(frmH.*w,nfft));
%Undo the logarithmic operation
fbout = (10.^((frme(1:nfft/2+1,:))/20));
fbFout = (10.^((frmeF(1:nfft/2+1,:))/20));
fbHout = (10.^((frmeH(1:nfft/2+1,:))/20));

%fbFHout = fbFout.*fbHout;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Heuristic final Filtering step...just seems to work/sound better...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fbCout = fbout - (mixFH*fbFout); %A Cleaned Version
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get the phase from the original input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bout = fe(1:nfft/2+1,:); %Original Spectrogram
pbout = angle(bout); %Phase

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Add back the original phase from the input...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fbout = abs(fbout).*exp(pbout*sqrt(-1));%Output without
fbFout = abs(fbFout).*exp(pbout*sqrt(-1));%Formant Envelope
fbHout = abs(fbHout).*exp(pbout*sqrt(-1));%Exitation (Pitch/White Noise)
fbCout = abs(fbCout).*exp(pbout*sqrt(-1));%Cleaned

%fbFHout = abs(fbFHout).*exp(pbout*sqrt(-1));%Formant Env .* Harm Env

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Invert the Spectrogram's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xout = ispecgram(fbout,nfft,fs,win,ov);
xout = xout / max(abs(xout(:)));            %Output witout mix subtraction
xFout = ispecgram(fbFout,nfft,fs,win,ov);
xFout = xFout / max(abs(xFout(:)));         %Formant Component Output
xHout = ispecgram(fbHout,nfft,fs,win,ov);
xHout = xHout / max(abs(xHout(:)));         %Harmonic Component Output
xCout = ispecgram(fbCout,nfft,fs,win,ov);
xCout = xCout / max(abs(xCout(:)));         %Enhanced Signal

%xFHout = ispecgram(fbFHout,nfft,fs,win,ov);
%xFHout = xFHout / max(abs(xFHout(:)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Display output if Necessary...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if PRINT
    %Display
    subplot(2,1,1),
    imagesc(t,f,20*log10(abs(bout)+eps));axis xy; title('Input');
    ca = caxis; colorbar;
    %caxis([-60,0]); colorbar;

    subplot(2,1,2),
    imagesc(t,f,20*log10(abs(fbCout)+eps)); axis xy;  title('fbCout');
    caxis([ca(1),ca(2)]); colorbar;
    %caxis([-60,0]); colorbar;
    drawnow;

    disp('Press a key to play input wav file');
    pause();
    wavplay(x,fs);
    disp('Press a key to play Reconstructed - w*G wav file');
    pause();
    wavplay(xCout,fs);
end

%Other Debugging output...
%{
subplot(4,1,3),
imagesc(t,f,20*log10(abs(fbFHout)+eps));axis xy; title('fbFout.*fbHout');
caxis([ca(1),ca(2)]); colorbar;
%caxis([-60,0]); colorbar;

subplot(4,1,4),
imagesc(t,f,20*log10(abs(fbFout)+eps)); axis xy;  title('fBGout');
caxis([ca(1),ca(2)]); colorbar;
%caxis([-60,0]); colorbar;
%}
```

```
%}
```

A.6.3 Speech/Non-Speech detection

```
function [Voicing, VoicingMax,timeLbls] = voicingDetector(in, fs, PRINT, ...
        mAvgPwr, DNWRD_EXPANSION,nfft,win,ov);
%Basically is a downward expanding speech/non-speech detector that uses the
%power spectrum of the signal to label sections as speech or non-speech.
%
% Eventually want to include voiced/unvoiced detection as well using the
% LPC residual or cepstral analysis, but that is currently not perfected
% and is left for future work.  For right now this uses just a power based
% threshold and the optional downward (forward+backward) expansion.
%
% INPUTS:
% in        - input mono audio signal vector
% fs        - sampling frequency.  If fs==[] or nargin==1, the algorithm
%               assumes that the input <IN> is already a two dimensional
%               spectrogram, otherwise the algorithm treats <IN> as and
%               audio signal vector and uses the specgram() function from
%               the signal processing toolbox to estimate the short time
%               fourier transform of the signal with parameters
%               <fs>,<nfft>,<win>, and <ov>.
% PRINT     - [(0)|1] Weather or not to show output.
% mAvgPwr   - (.00056234 or .01) Speech/non-speech threshold level.
% DNWRD_EXPANSION -[0|(1)] Turn downward Expasion Off/On
% nfft      - (256) NFFT to use in call to specgram()
% win       - (=nfft) WIN to use in call to specgram()
% ov        - (=round(win*.80)) OV to use in call to specgram()
%
% OUTPUTS:
% Voicing  - is an array the same size as the number of frames in the
%           spectrogram with ones representing speech frames and 0's
%           representing non speech.  (When voicing detection is finally
%           added, voiced frames will == 2, unvoiced == 1, and silence ==
%           0)
%
% VoicingMax - is 1 only at locations that are a local maximum of the power
%           signature
% timeLbls  - is the time labels for each frame
%
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com


if nargin < 3
    PRINT = 0;
end

if nargin < 4
    if (nargin < 5 || DNWRD_EXPANSION)
        mAvgPwr = .01;
    else
        mAvgPwr = .00056234;
    end
end

if nargin < 5
    DNWRD_EXPANSION = 1;
end

if nargin < 6
```

```
    nfft = 256;
end
if nargin <7
    win = nfft;
end
if nargin <8
    ov = round(win*.80);
end

if ( (nargin == 1) || isempty(fs) )
    b1 = in;
    freqLbls = 1:size(b1,1);
    timeLbls = 1:size(b1,2);
else
    MAXFS = 8000;
    if MAXFS < fs
        in = resample(in, MAXFS, fs);
        in = in(:)';
        in = in/max(abs(in(:)));
    end
    [b1,freqLbls,timeLbls] = specgram(in,nfft,MAXFS,win,ov);
end

nfft = (size(b1,1)-1)*2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Find those frames that have enough signal power to justify.  We start by
%finding the Average power for each frame.  We then find the frames that
%are a local maximum and are above a threshold.  We then use downward
%expansion to expand those peaks down to their connecting local minimum.
%That way, even if the onset of the speech segment is below the threshold
%power level, we should be able to still capture it.

%calculate short time signal power...
stp(:,1) = abs(b1(:,1)).^2;
AP = mean(abs(b1).^2);

%low pass filter the power signature...
[b,a] = butter(3,1/2,'low');
AvgPwr = filter(b,a,AP);
AvgPwr = AvgPwr / max(AvgPwr(:));

lmnAvgPwr = localMin(AvgPwr);
lmxAvgPwr = localmax(AvgPwr);

%Initialize
Voicing = AvgPwr(1) > mAvgPwr;

%Find Max Voicing Positions
VoicingMax = lmxAvgPwr.*AvgPwr > mAvgPwr;

if DNWRD_EXPANSION
    %Go Forwards and backwards across the frames to find the points where the
    %power is above the threshold, and expand those regions out to their
    %nearest local minima.  Acts as a sort of downward expanding noise
    %gate.

    %Forward
    for i = 2:size(b1,2)
        if Voicing(i-1) == 0
            Voicing(i) = AvgPwr(i) > mAvgPwr;
        else
            if (lmnAvgPwr(i)==1 && AvgPwr(i) < mAvgPwr)
                Voicing(i) = 0;
            else
                Voicing(i) = 1;
            end
        end
```

```
    end

    %
    %Backward
    for i = size(b1,2)-1:-1:2
        if Voicing(i+1) == 0
        else
            if (lmnAvgPwr(i)==1 && AvgPwr(i) < mAvgPwr)
                Voicing(i) = 0;
            else
                Voicing(i) = 1;
            end
        end
    end
    %}

else
    %Just use Simple Thresholding
    Voicing = AvgPwr > mAvgPwr;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUTURE WORK:
%Would like to add using the real Cepstrum to determine if each of the
%frames with significant power are voiced or un-voiced.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Forward-pass filter to remove sporadic false negatives
for i = 2:size(b1,2)-1
    if Voicing(i-1) == Voicing(i+1)
        Voicing(i) = Voicing(i-1);
    end
end

%Display output...
if (nargout == 0 || PRINT ==1)
    hold off;
    subplot(2,1,1), imagesc(20*log10(abs(b1)+eps)); axis tight; axis xy;
    hold on; plot(Voicing*10, 'k'); hold off;
    subplot(2,1,2), plot(20*log10(abs(AvgPwr+eps))); axis tight; hold on;
    plot(20*log10(abs(ones(size(AvgPwr))*mAvgPwr)), 'r:'); axis tight;
    stem(20*log10(Voicing+eps), 'k:', 'Marker', 'None'); axis tight;
    hold off;
    drawnow;
end
```

A.6.4 Vocal Onset Point Detection

```
function [VOP_Times, VOP, dVOP] = VocalOnsetPoints(m, fs, win, ov, NCoeff)
%Find the vocal onset points for a mono speech signal <m> with sampling
%frequency <fs> using window <win>, overlap <ov>, and using <NCoeff> LPC
%Coefficients.
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

if nargin < 6
    PRINT = 0;
end

NCoeff = 64;
NSamples = length(m);
```

```
NSeconds = NSamples / fs;

mix = makeframes(m,win,ov,'hamming');
NFrames = size(mix,2);
FrameLabels = ((1:NFrames)/NFrames)*NSeconds;

[a,g] = lpc(mix,NCoeff);                %Get LPC Coeffs

est_x = zeros(size(mix));

for i = 1:NFrames
    est_x(:,i) = filter([0 -a(i, 2:end)],1,mix(:,i)); %Estimate the Signal
end

e = mix-est_x;                      %Residual

EnS = sum(abs(mix.^2));     %Energy of Signal
EnR = sum(abs(e.^2));       %Energy of Residual

VOP = EnS./EnR;


dVOP = (circshift(VOP,[0 -1])-circshift(VOP,[0 1]));

%Should we LPF?
[B,A] = butter(3,.09,'low');
dVOP = filtfilt(B,A,dVOP);

dVOP = dVOP / max(dVOP(:));
dVOP(find(dVOP < .1)) = 0;

lmxdVOP = localmax(dVOP);
VOPr = lmxdVOP.*dVOP;
VOP_Times = lmxdVOP.*FrameLabels;
VOP_Times(find(VOP_Times == 0)) = [];

if (PRINT || nargout == 0)
    b = 20*log10(abs(fft(mix,256))+eps);
    b = b(1:127,:);
    imagesc(b), axis xy; hold on;
    %plot(127*dVOP/max(abs(dVOP(:)))); axis tight;
    stem(127*VOPr / max(abs(VOPr(:))),'k','Marker','None','LineWidth',4); hold off;
    drawnow;
end
```

A.6.5 KL Divergence

```
function out = KL(gmm1,gmm2,x)
%Get the approximate asymetric KL-divergence between two GMM's using
%the feature vectors contained in x
%
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

out = 0;
if ((nargin < 3) || (length(x)==1))
    if nargin < 3
        SampSize = 5000; %Default Sample Size
    else
        SampSize = x;
    end
    x = gmmsamp(gmm1,SampSize);
```

```
else

fx = gmmprob(gmm1,x);
gx = gmmprob(gmm2,x);
out = 1/SampSize*sum(log(fx./gx));
```

A.6.6 Dividing Data into Frames

```
function [Y, nrows, ncol] = makeframes(varargin)
%FUNCTION makeframes --
%   Separates a mono input signal into a set of frames.  Output matrix is a
%   MxN.  Where M = the frame length, and sequential frames are stored in
%   the N columns.  To return to a mono signal use the command line
%   >> orig = y(:);
%
%   This function is used to make sure that all processes on an input
%   signal that require the signal to be separated into frames will use
%   consistently framed data so that input and output lengths will match.
%
%   Input Arguments:  [SIG, FRAMELEN, OVERLAP, WINFLAG]
%   SIG - the input signal
%   FRAMELEN - the length of each frame, in samples
%   OVERLAP - the amount of overlap between frames, in samples
%       (default 0)
%   WINFLAG - Type of window to apply to the data.  Use 'hamming' for a
%       hamming window.  Use 'none' for no window applied to the
%       data.  'none' is the default.
%
%Adapted from the specgram.m code by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

%

error(nargchk(1,4,nargin))
[msg,x,winlen,noverlap,wflag]=chk(varargin);
error(msg)

x = x/max(x(:));

nx = length(x);
nwind = winlen;
if nx < nwind    % zero-pad x if it has length less than the window length
    x(nwind)=0;  nx=nwind;
end
x = x(:); % make a column vector for ease later
ncol = fix((nx-noverlap)/(nwind-noverlap));
colindex = 1 + (0:(ncol-1))*(nwind-noverlap);
rowindex = (1:nwind)';
if length(x)<(nwind+colindex(ncol)-1)
    x(nwind+colindex(ncol)-1) = 0;   % zero-pad x
end

Y = zeros(nwind,ncol);
nrows = nwind;
Y(:) = x(rowindex(:,ones(1,ncol))+colindex(ones(nwind,1),:)-1);

%If we need to use a hamming window
if strcmp(wflag, 'hamming')
    [M,N] = size(Y);
    w = hamming(M); % hamming window
    w = w(:, ones(1,N)); %make same size as output
    Y = Y.*w; %apply window to data
end
```

```
function [msg,x,winlen,noverlap,wflag] = chk(P)
%Parse the varargin values for use in the function
%
msg = [];
x = P{1};

if (length(P) > 1) & ~isempty(P{2})
    %winlen = P{3}*Fs/1000;
    winlen = P{2};
else
    winlen = 256;
end

if length(P) > 2 & ~isempty(P{3})
    noverlap = P{3};
else
    noverlap = 0;
end

if length(P) > 3 & ~isempty(P{4})
    wflag = P{4};
else
    wflag = 'none';
end

% NOW do error checking
if min(size(x))~=1,
    msg = 'Requires vector (either row or column) input.';
end
```

A.6.7 Removing Duplicates From an Array

```
function out = removeDuplicates(a)
%Remove duplicate values from an array
%
%Written by:
%Jonathan Lareau - Rochester Insititute of Technology - 2006
%programming@jonlareau.com

out = [];
for i = 1:length(a)
    if isempty(find(out==a(i)))
        out(end+1) = a(i);
    end
end
```

Appendix B

Third Party Software

B.1 Full Packages

B.1.1 NETLAB[18]

Original Author: Ian Nabney and Christopher Bishop

Available at: http://www.ncrg.aston.ac.uk/netlab/index.php

B.1.2 RASTA-MAT[4]

Original Author: Dan Ellis

Available at: http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/

B.1.2.1 Select Changes Made to the RASTA-MAT Package

In the course of developing this software it became necessary to make minor changes to the RASTA-MAT package. These changes relate to the parameter passing methodology and were required to guarantee consistency in feature vector calculation. The modified version of the rastaplp.m file is included below.

B.1.2.2 rastaplp.m

```
function [cepstra, spectra, pspectrum, lpcas, F, M] = rastaplp(samples, sr, dorasta, ...
     modelorder, win, ov, lifterexp)
%[cepstra, spectra, lpcas] = rastaplp(samples, sr, dorasta, modelorder, win, ov, lifterexp)
```

```
%
% cheap version of log rasta with fixed parameters
%
% output is matrix of features, row = feature, col = frame
%
% sr is sampling rate of samples, defaults to 8000
% dorasta defaults to 1; if 0, just calculate PLP
% modelorder is order of PLP model, defaults to 8.  0 -> no PLP
%
% rastaplp(d, sr, 0, 12) is pretty close to the unix command line
% feacalc -dith -delta 0 -ras no -plp 12 -dom cep ...
% except during very quiet areas, where our approach of adding noise
% in the time domain is different from rasta's approach
%
% 2003-04-12 dpwe@ee.columbia.edu after shire@icsi.berkeley.edu's version
%
%Modified 2006 by Jonathan Lareau to allow for LIFTEREXP, WIN, & OV as inputs

if nargin < 2
  sr = 8000;
end
if nargin < 3
  dorasta = 1;
end
if nargin < 4
  modelorder = 8;
end
if nargin <5
    win = 256
end
if nargin < 6
    ov = win / 2;
end
if nargin < 7
    lifterexp = .6;
end

% add miniscule amount of noise
%samples = samples + randn(size(samples))*0.0001;

% first compute power spectrum
pspectrum = powspec(samples, sr, win/sr, (win-ov)/sr);

% next group to critical bands
aspectrum = audspec(pspectrum, sr);
nbands = size(aspectrum,1);

if dorasta ~= 0

  % put in log domain
  nl_aspectrum = log(aspectrum);

  % next do rasta filtering
  ras_nl_aspectrum = rastafilt(nl_aspectrum);

  % do inverse log
  aspectrum = exp(ras_nl_aspectrum);

end

% do final auditory compressions
postspectrum = postaud(aspectrum, sr);

if modelorder > 0

  % LPC analysis
  lpcas = dolpc(postspectrum, modelorder);
```

```
  % convert lpc to cepstra
  cepstra = lpc2cep(lpcas, modelorder+1);

  % .. or to spectra
  [spectra,F,M] = lpc2spec(lpcas, nbands);

else

  % No LPC smoothing of spectrum
  spectra = postspectrum;
  cepstra = spec2cep(spectra);

end

cepstra = lifter(cepstra, lifterexp);
```

B.2 Individual m-files

B.2.1 Orderby.m

Original Author: Sara Silvia

Available at: http://www.mathworks.com/matlabcentral/fileexchange/

```
function ox=orderby(x,i,varargin)
%ORDERBY    Orders vectors and matrices according to a predefined order.
%   ORDERBY(X,I) orders the elements of a vector or matrix X
%   according to the index matrix I.
%
%   ORDERBY(X,I,DIM) orders along the dimension DIM.
%
%   If X is a vector, then Y = X(I). If X is an m-by-n matrix, then
%       for j = 1:n, Y(:,j) = X(I(:,j),j); end
%
%   Input arguments:
%      X - the vector or matrix to order (array)
%      I - the index matrix with the ordering to apply (array)
%      DIM - the dimension along which to order (integer)
%   Output arguments:
%      Y - the ordered vector or matrix (array)
%
%   Examples:
%      X = [10 25 30 40]
%      I = [3 2 1 4]
%      Y = ORDERBY(X,I)
%      Y = 30    25    10    40
%
%      X = [10 25 ; 3.2 4.1 ; 102 600]
%      I = [2 3 ; 1 1 ; 3 2]
%      Y = ORDERBY(X,I)
%      Y =   3.2000  600.0000
%           10.0000   25.0000
%          102.0000    4.1000
%
%      X = [10 25 50 ; 3.2 4.1 5.5 ; 102 600 455 ; 0.03 0.34 0.01]
%      I = [1 4 3 2]
%      DIM = 1
%      Y = ORDERBY(X,I,DIM)
%      Y =  10.0000   25.0000   50.0000
%            0.0300    0.3400    0.0100
```

```
%          102.0000  600.0000  455.0000
%            3.2000    4.1000    5.5000
%
%      X = [10 25 50 ; 3.2 4.1 5.5 ; 102 600 455 ; 0.03 0.34 0.01]
%      I = [1 3 2]
%      DIM = 2
%      Y = ORDERBY(X,I,DIM)
%      Y =  10.0000   50.0000   25.0000
%             3.2000    5.5000    4.1000
%          102.0000  455.0000  600.0000
%            0.0300    0.0100    0.3400
%
%   See also SHUFFLE
%
%   Created: Sara Silva (sara@itqb.unl.pt) - 2002.11.02

if size(x,1)==1 | size(x,2)==1
   % if its a vector, we don't even care about the eventual 3rd argument (dim)
        ox=x(i);
else
   % if its a matrix, lets check if the ordering is along one dimension
   switch nargin
   case 2
      % each column is ordered separately
        for c=1:size(x,2)
        ox(:,c)=x(i(:,c),c);
      end
   case 3
      % let's see if we're ordering entire rows or cols
        d=varargin{1};
        switch d
      case 1
         % rows
         ox=x(i,:);
      case 2
         % cols
         ox=x(:,i);
        otherwise
        error('ORDERBY: Unknown command option.')
      end
   end
end
```

B.2.2 Shuffle.m

Original Author: Sara Silvia

Available at: http://www.mathworks.com/matlabcentral/fileexchange/

```
function [s,myorder]=shuffle(x,varargin)
%SHUFFLE    Shuffles vectors or matrices.
%   SHUFFLE(X) shuffles the elements of a vector or matrix X.
%
%   SHUFFLE(X,DIM) shuffles along the dimension DIM.
%
%   [Y,I] = SHUFFLE(X) also returns an index matrix I. If X is
%   a vector, then Y = X(I). If X is an m-by-n matrix, then
%       for j = 1:n, Y(:,j) = X(I(:,j),j); end
%
%   Input arguments:
%      X - the vector or matrix to shuffle (array)
%      DIM - the dimension along which to shuffle (integer)
```

```
%   Output arguments:
%      Y - the vector or matrix with the elements shuffled (array)
%      I - the index matrix with the shuffle order (array)
%
%   Examples:
%      X = [10 25 30 40]
%      [Y,I] = SHUFFLE(X)
%      Y = 30    25    10    40
%      I =  3     2     1     4
%
%      X = [10 25 ; 3.2 4.1 ; 102 600]
%      [Y,I] = SHUFFLE(X)
%      Y =   3.2000  600.0000
%           10.0000   25.0000
%          102.0000    4.1000
%      I =  2     3
%           1     1
%           3     2
%
%      X = [10 25 50 ; 3.2 4.1 5.5 ; 102 600 455 ; 0.03 0.34 0.01]
%      DIM = 1
%      [Y,I] = SHUFFLE(X,DIM)
%      Y =  10.0000   25.0000   50.0000
%            0.0300    0.3400    0.0100
%          102.0000  600.0000  455.0000
%            3.2000    4.1000    5.5000
%      I =  1     4     3     2
%
%      X = [10 25 50 ; 3.2 4.1 5.5 ; 102 600 455 ; 0.03 0.34 0.01]
%      DIM = 2
%      [Y,I] = SHUFFLE(X,DIM)
%      Y =  10.0000   50.0000   25.0000
%            3.2000    5.5000    4.1000
%          102.0000  455.0000  600.0000
%            0.0300    0.0100    0.3400
%      I =  1     3     2
%
%   See also ORDERBY
%
%   Created: Sara Silva (sara@itqb.unl.pt) - 2002.11.02

rand('state',sum(100*clock)); % (see help RAND)

switch nargin
case 1
   if size(x,1)==1 | size(x,2)==1
      [ans,myorder]=sort(rand(1,length(x)));
      s=x(myorder);
   else
      [ans,myorder]=sort(rand(size(x,1),size(x,2)));
        for c=1:size(x,2)
        s(:,c)=x(myorder(:,c),c);
      end
   end
case 2
   d=varargin{1};
   switch d
   case 1
      [ans,myorder]=sort(rand(1,size(x,1)));
                s=x(myorder,:);
   case 2
      [ans,myorder]=sort(rand(1,size(x,2)));
      s=x(:,myorder);
   otherwise
      error('SHUFFLE: Unknown command option.')
   end
end
```

B.2.3 localmax.m

Original Author: Duane Hanselman

Available at: http://www.mathworks.com/matlabcentral/fileexchange/

```
function [bx,by]=localmax(y)
%LOCALMAX(Y) Local Maxima, Peak Detection.
% LOCALMAX(Y) when Y is a vector returns a logical vector the same size as
% Y containing logical True where the corresponding Y value is a local
% maximum, that is where Y(k-1)<Y(k)>Y(k+1).
%
% LOCALMAX(Y) when Y is a matrix returns a logical matrix the same size as
% Y containing logical True where the corresponding Y value is a
% local maxima down each column, i.e., Y(k-1,n)<Y(k,n)>Y(k+1,n).
%
% [BX,BY] = LOCALMAX(Z) when Z is a matrix returns two logical matrices the
% same size as Z containing logical True where the corresponding Z value is
% a logical maxima. BX identifies the maxima across each row, i.e.,
% Z(k,n-1)<Z(k,n)>Z(k,n+1), and BY identifies the local maxima down each
% column, i.e., Z(k-1,n)<Z(k,n)>Z(k+1,n).
%
% When two or more consecutive data points have the same local maxima
% value, the last one is identified. First and last data points are
% returned if appropriate.
%
% See also MAX.

% D.C. Hanselman, University of Maine, Orono, ME 04469
% MasteringMatlab@yahoo.com
% Mastering MATLAB 7
% 2005-12-05

if ~isreal(y)
   error('Y Must Contain Real Values Only.')
end
ry=size(y,1);
isrow=ry==1;
if isrow    % convert to column for now
   y=y(:);
end
if nargout==2
   if isvector(y)
      error('Second Output Argument Not Needed.')
   end
   by=local_getmax(y);
   bx=local_getmax(y')';
else  % one output
   bx=local_getmax(y);
   if isrow
   bx=bx.';
   end
end
%--------------------------------------------------------------------------
function b=local_getmax(y)
infy=-inf(1,size(y,2));
k=sign(diff([infy; y; infy]));
b=logical(diff(k+(k==0))==-2);
```

B.2.4 localmin.m

Original Author: Duane Hanselman

Adapted by Jonathan Lareau from localmax.m

```
function [bx,by]=localmin(y)
%LOCALMIN(Y) Local Maxima, Peak Detection.
% LOCALMIN(Y) when Y is a vector returns a logical vector the same size as
% Y containing logical True where the corresponding Y value is a local
% maximum, that is where Y(k-1)>Y(k)<Y(k+1).
%
% LOCALMIN(Y) when Y is a matrix returns a logical matrix the same size as
% Y containing logical True where the corresponding Y value is a
% local maxima down each column, i.e., Y(k-1,n)?Y(k,n)<Y(k+1,n).
%
% [BX,BY] = LOCALMIN(Z) when Z is a matrix returns two logical matrices the
% same size as Z containing logical True where the corresponding Z value is
% a logical maxima. BX identifies the minima across each row, i.e.,
% Z(k,n-1)>Z(k,n)<Z(k,n+1), and BY identifies the local minima down each
% column, i.e., Z(k-1,n)>Z(k,n)<Z(k+1,n).
%
% When two or more consecutive data points have the same local minima
% value, the last one is identified. First and last data points are
% returned if appropriate.
%

% D.C. Hanselman, University of Maine, Orono, ME 04469
% MasteringMatlab@yahoo.com
% Mastering MATLAB 7
% 2005-12-05
%
%Adapted from localmax.m by Jonathan Lareau - RIT - 2006

if ~isreal(y)
   error('Y Must Contain Real Values Only.')
end
ry=size(y,1);
isrow=ry==1;
if isrow    % convert to column for now
   y=y(:);
end
if nargout==2
   if isvector(y)
      error('Second Output Argument Not Needed.')
   end
   by=local_getmin(y);
   bx=local_getmin(y')';
else  % one output
   bx=local_getmin(y);
   if isrow
   bx=bx.';
   end
end
%-------------------------------------------------------------------------
function b=local_getmin(y)
infy=-inf(1,size(y,2));
k=sign(diff([infy; y; infy]));
b=logical(diff(k+(k==0))==2);
```

B.2.5 nearestpoint.m

Original Author: Jos vander Geest

Available at: http://www.mathworks.com/matlabcentral/fileexchange/

```
function [IND, D] = nearestpoint(x,y,m) ;
% NEARESTPOINT - find the nearest value in another vector
%
%   IND = NEARESTPOINT(X,Y) finds the value in Y which is the closest to
%   each value in X, so that abs(Xi-Yk) => abs(Xi-Yj) when k is not equal to j.
%   IND contains the indices of each of these points.
%   Example:
%      NEARESTPOINT([1 4 12],[0 3]) -> [1 2 2]
%
%   [IND,D] = ... also returns the absolute distances in D,
%   that is D == abs(X - Y(IND))
%
%   NEARESTPOINT(X, Y, M) specifies the operation mode M:
%   'nearest' : default, same as above
%   'previous': find the points in Y that just precedes a point in X
%               NEARESTPOINT([1 4 12],[0 3],'previous') -> [1 1 1]
%   'next'    : find the points in Y that directly follow a point in X
%               NEARESTPOINT([1 4 12],[0 3],'next') -> [2 NaN NaN]
%
%   If there is no previous or next point in Y for a point X(i), IND(i)
%   will be NaN.
%
%   X and Y may be unsorted.
%
%   This function is quite fast, and especially suited for large arrays with
%   time data. For instance, X and Y may be the times of two separate events,
%   like simple and complex spike data of a neurophysiological study.
%
%

%   Nearestpoint('test') will run a test to show it's effective ness for
%   large data sets

% Created       : august 2004
% Author        : Jos van der Geest
% Email         : matlab@jasen.nl
% Modifications :
%  aug 25, 2004 - corrected to work with unsorted input values
%  nov 02, 2005 -

if nargin==1 & strcmp(x,'test'),

    testnearestpoint ;
    return
end

error(nargchk(2,3,nargin)) ;

if nargin==2,
    m = 'nearest' ;
else
    if ~ischar(m),
        error('Mode argument should be a string (either ''nearest'', ''previous'', or ''next'')') ;
    end
end

if ~isa(x,'double') | ~isa(y,'double'),
    error('X and Y should be double matrices') ;
end

% sort the input vectors
sz = size(x) ;
[x, xi] = sort(x(:)) ;
[dum, xi] = sort(xi) ; % for rearranging the output back to X
nx = numel(x) ;
cx = zeros(nx,1) ;
qx = isnan(x) ; % for replacing NaNs with NaNs later on
```

```
[y,yi] = sort(y(:)) ;
ny = length(y) ;
cy = ones(ny,1) ;

xy = [x ; y] ;

[xy, xyi] = sort(xy) ;
cxy = [cx ; cy] ;
cxy = cxy(xyi) ; % cxy(i) = 0 -> xy(i) belongs to X, = 1 -> xy(i) belongs to Y
ii = cumsum(cxy) ;
ii = ii(cxy==0).' ; % ii should be a row vector

% reduce overhead
clear cxy xy xyi ;

switch lower(m),
    case {'nearest','near','absolute'}
        % the indeces of the nearest point
        ii = [ii ; ii+1] ;
        ii(ii==0) = 1 ;
        ii(ii>ny) = ny ;
        yy = y(ii) ;
        dy = abs(repmat(x.',2,1) - yy) ;
        [dum, ai] = min(dy) ;
        IND = ii(sub2ind(size(ii),ai,1:nx)) ;
    case {'previous','prev','before'}
        % the indices of the previous points
        ii = [ii(2:end) ii(end)] ;
        ii(ii < 1) = NaN ;
        IND = ii ;
    case {'next','after'}
        % the indices of the next points
        ii = ii + 1 ;
        ii(ii>ny) = NaN ;
        IND = ii ;
    otherwise
        error(sprintf('Unknown method "%s"',m)) ;
end

IND(qx) = NaN ; % put NaNs back in

if nargout==2,
    % also return distance if requested;
    D = repmat(NaN,1,nx) ;
    q = ~isnan(IND) ;
    D(q) = abs(x(q) - y(IND(q))) ;
    D = reshape(D(xi),sz) ;
end

% reshape and sort to match input X
IND = reshape(IND(xi),sz) ;

% because Y was sorted, we have to unsort the indices
q = ~isnan(IND) ;
IND(q) = yi(IND(q)) ;

% END OF FUNCTION

function testnearestpoint
disp('TEST for nearestpoint, please wait ... ') ;
M = 13 ;
tim = repmat(NaN,M,3) ;
tim(8:M,1) = 2.^[8:M].' ;
figure('Name','NearestPointTest','doublebuffer','on') ;
h = plot(tim(:,1),tim(:,2),'bo-',tim(:,1),tim(:,3),'rs-') ;
xlabel('N') ;
ylabel('Time (seconds)') ;
```

```
title('Test for Nearestpoint function ... please wait ...') ;
set(gca,'xlim',[0 max(tim(:,1))+10]) ;
for j=8:M,
    N = 2.^j ;
    A = rand(N,1) ; B = rand(N,1) ;
    tic ;
    D1 = zeros(N,1) ;
    I1 = zeros(N,1) ;
    for i=1:N,
        [D1(i), I1(i)] = min(abs(A(i)-B)) ;
    end
    tim(j,2) = toc ;
    pause(0.1) ;
    tic ;
    [I2,D2] = nearestpoint(A,B) ;
    tim(j,3) = toc ;
    % isequal(I1,I2)
    set(h(1),'Ydata',tim(:,2)) ;
    set(h(2),'Ydata',tim(:,3)) ;
    drawnow ;
end
title('Test for Nearestpoint function') ;
legend('Traditional for-loop','Nearestpoint',2) ;
```