Fall 2013

Mispronunciation Detection for Language Learning and Speech Recognition Adaptation

Zhenhao Ge
Purdue University

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Zhenhao Ge

Entitled

Mispronunciation Detection for Language Learning and Speech Recognition Adaptation

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

MARK J. SMITH
Chair

EDWARD J. DELP

MARY L. COMER

XIN LUO

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): MARK J. SMITH

Approved by: M. R. Melloch 12-05-2013
Head of the Graduate Program Date

MISPRONUNCIATION DETECTION FOR LANGUAGE LEARNING AND SPEECH RECOGNITION ADAPTATION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Zhenhao Ge

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2013

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

Foremost, I would like to express my deepest gratitude to my academic advisor Prof. Mark J.T. Smith, for his continuous support and providing me with an excellent atmosphere for my Ph.D study and research, for his patience, motivation, enthusiasm and immense knowledge. I could not have imagined having a better advisor and mentor to guide me through the research and writing of this thesis.

Besides my advisor, I would like to sincerely thank the rest of my thesis committee members: Prof. Edward J. Delp, Prof. Mary Comer and Prof. Xin Luo. Without their insightful comments and consistent encouragement, conducting research and writing the thesis could not be such a beneficial and joyful experience.

Next, I would like to address my sincere thanks to the members of speech analytics team of Interactive Intelligence, where I worked as software engineer intern for about nine months cumulatively. I would like to thank my supervisor Aravind Ganapathiraju, who shared lots of ideas and thoughts through the implementation of pronunciation learning and accent classification, I could not have imagined finishing the research of mispronunciation detection for speech recognition adaptation without such a knowledgeable and skillful expert in speech recognition.

I would also like to thank my mentor Ananth Iyer, with whom I had many fruitful discussions through the implementation of pronunciation learning algorithm. I earned a lot of hands-on experience from Ananth, to tune the acoustic and language models. He also provided me all the necessary resources for my research and experiments during my intern in ININ, such as ININ phoneme recognizer, lexicon tester, etc.

I would also like to thank Vivek Tygi, for kindly sharing with me his setup of speech recognition system, for his suggestions in pronunciation pruning algorithm design and for his help in training HLDA features for accent classification; thank Yingyi Tan, for her patient mentoring and critical comments on accent classification

implementation; thank Scott Randal for providing me support and suggestions in pronunciation learning and other projects, as a computational linguists; thank Kevin Vlack for his strictness on coding and his suggestions to make improvement in code readability and efficiency; thank all other team members for providing me such a nice and productive working environment. I want to also extend my thanks to Felix Wyss, the leader of media group in ININ, without his and Aravind's approval, I would not have this wonderful intern experience and step forward in the research of speech.

Further, I would like to thank my supervisor Jian Cheng, my mentor Xin Chen and senior scientist and consultant Jared Bernstein in Pearson Knowledge-Technologies, where I worked as research intern on speech recognition with Deep Neural Networks. Though the research with the company is not covered under this thesis, they provided me lots of meaningful suggestions and ideas through reviewing my thesis work, such as testing the algorithms with multiple larger data sets and the utilize Restricted Boltzmann Machine in discriminative training of accents.

In addition, I would like to thank my advisor Prof. Paul Salama and co-advisor Prof.Mohamed El-Sharkawy during my master study at IUPUI, who enlightened me the first glance of speech processing, and also thank the speech research founder at that time, Dr. Charles S. Watson and Dr. James D. Miller from Communication Disorder Technologies (CDT), for assigning me the work of pronunciation evaluation for American-accented Arabics, which started my research in speech.

Moreoever, I would like to thank my labmate Sudhendu Raj Sharma, for his great support and corporation as research buddy, and also as personal good friend. I would also like to thank my classmates and friends at Purdue, Xujie Zhang, Yandong Guo, Zhi Zhou, Yi Fang, Zhou Yu, Ruoqiao Zhang, etc., for their forever friendship and giving me precious memories of study and research at Purdue.

Last but not the least, I would like to thank my parents Jincai Ge and Xinhong Shen, for giving me birth in the first place and supporting me spiritually throughout my life. I would also like to thank my considerate wife Yufang Sun, her support, encouragement, quiet patience and unwavering love are undeniabe.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AFCCs	Adaptive Frequency Cepstral Coefficients
ASR	Automated Speech Recognition
CALL	Computer Assisted Language Learning
CAPT	Computer Assisted Pronunciation Training
CMU	Carnegie Mellon University
CMVN	Cepstral Mean and Variance Normalization
DCT	Discrete Cosine Transform
DNN	Deep Neural Networks
DTW	Dynamic Time Warping
EER	Equal Error Rate
FAR	False Acceptance Rate
FLL	Foreign Language Learning
FRR	False Rejection Rate
GOP	Goodness of Pronunciation
HLDA	Heteroscedastic Linear Discriminant Analysis
HMM	Hidden Markov Models
HTK	Hidden Markov Model Toolkit
LDA	Linear Discriminative Analysis
LDC	Linguistic Data Consortium
ININ	Interactive Intelligence
LLR	Log-Likelihood Ratio
LM	Language Model
LOO	Leave One Out
LRT	Likelihood Ratio Test

LSA	Latent Semantic Analysis
LVCSR	Large Vocabulary Continuous Speech Recognition
IVR	Interactive Voice Response
MAP	Maximum a Posterior
MFC	Mel Frequency Cepstrum
MFCCs	Mel Frequency Cepstral Coefficients
MLE	Maximum Likelihood Training
MLLR	Maximum Likelihood Linear Regression
MVN	Mean and Variance Normalization
NER	Name Error Rate
NIST	National Institute of Standards and Technology
PCA	Principle Component Analysis
PLP	Perceptual Linear Predictive
PSM	Pronunciation Space Models
RBM	Restricted Boltzmann Machine
SD	Steepest Descent
SER	Sentence Error Rate
SRI	Stanford Research Institute
STFT	Short Time Fourier Transform
SVM	Support Vector Machine
TCS	Total Confidence Score
TIMIT	Corpus of American English with different sexes and dialects developed by Texas Instruments (TI) and Massachusetts Institute of Technology (MIT)
VQ	Vector Quantization
WCS	Word Confidence Score
WER	Word Error Rate

ABSTRACT

Ge, Zhenhao Ph.D., Purdue University, December 2013. Mispronunciation Detection for Language Learning and Speech Recognition Adaptation. Major Professor: Mark J.T. Smith.

The areas of "mispronunciation detection" (or "accent detection" more specifically) within the speech recognition community are receiving increased attention now. Two application areas, namely language learning and speech recognition adaptation, are largely driving this research interest and are the focal points of this work.

There are a number of Computer Aided Language Learning (CALL) systems with Computer Aided Pronunciation Training (CAPT) techniques that have been developed. In this thesis, a new HMM-based text-dependent mispronunciation system is introduced using text Adaptive Frequency Cepstral Coefficients (AFCCs). It is shown that this system outperforms the conventional HMM method based on Mel Frequency Cepstral Coefficients (MFCCs). In addition, a mispronunciation detection and classification algorithm based on Principle Component Analysis (PCA) is introduced to help language learners identify and correct their pronunciation errors at the word and syllable levels.

To improve speech recognition by adaptation, two projects have been explored. The first one improves name recognition by learning acceptable variations in name pronunciations, as one of the approaches to make grammar-based name recognition adaptive. The second project is accent detection by examining the shifting of fundamental vowels in accented speech. This approach uses both acoustic and phonetic information to detect accents and is shown to be beneficial with accented English. These applications can be integrated into an automated international calling system, to improve recognition of callers' names and speech. It determines the callers' accent

based in a short period of speech. Once the type of accents is detected, it switches from the standard speech recognition engine to an accent-adaptive one for better recognition results.

1. INTRODUCTION

Mispronunciation and accent detection based on computer techniques are receiving increased attention nowadays. Two applications namely language learning and speech recognition adaptation, contribute significantly to this research interest and are the focal points of this work. In both applications, adaptive features are used to enhance the performance of mispronunciation detections.

1.1 Applications in Language Learning

Millions of people throughout the world study foreign languages. However, many of them do not receive the one-on-one instruction needed to master proper pronunciations. To address this problem, automated computer tools are being developed to help language learners. Mispronunciation detection systems based on Computer-assisted Language Learning (CALL) [1] attempt to detect mistakes made by language learners at either the phone, word, or sentence level [2, 3], and inform the learner of those errors.

In this thesis, the goal of mispronunciation detection for language learning is to detect phone-level and syllable-level mispronunciations in words and sentences. A baseline text-dependent word mispronunciation detection system based on Hidden Markov Models (HMM) was constructed. Mispronunciations are detected by comparing the log likelihood of the potential state of the targeting pronunciation unit with a certain pre-determined threshold, given HMM model parameter λ and observation of the testing sample.

Although the emerging area of mispronunciation detection is relatively new, the general area of speech recognition is quite mature. This allow us to focus on and explore new adaptive features that can facilitate detection of accents and mispronuncia-

tions. One of the most commonly used features for speech recognition and evaluation systems are Mel Frequency Cepstral Coefficients (MFCCs), which are time-frequency signal parameters. In the process of generating these features, frequency bands are warped from the linear scale to the Mel scale. The Mel frequency scale models the sensitivity of the human auditory system, thereby allowing an algorithm that employs this scale to mimic the genetically evolved biologic process humans use to recognize words. The Mel scale warping function and the resulting MFCC have been used for many years now and are the gold standard for feature-based HMM recognition systems. In this thesis, we employ new adaptive features for mispronunciation detection in language learning with the goal of improving performance.

1.2 Applications in Speech Recognition Adaptation

As businesses become more global, speech recognition system used in industry will have to be able to accommodate callers from all over the world. Currently, this is a major challenge. Callers with accents are typically not served well by existing speech recognition system. Detecting mispronunciations can be used to improve the performance of speech recognition. This transform from standard speech recognition to language-adaptive [4] or speaker-adaptive [5] speech recognition is desired to better serve people with accents.

In this thesis, the goal of mispronunciation detection for speech recognition adaptation is to identify the types of mispronunciations, namely accents of speakers. To help evaluate the utility of the new methods proposed and their potential impact on society users, we have partnered with Interactive Intelligence Inc. (ININ). ININ has provided calling data and a test environment for assessment.

The big challenge in this application compared with language learning is that the transcription of the speech recorded during the short-time period is unavailable, and the only assumption we can make is the speaker should speak in English. So prior to accent detection, a subset of recognized speech with a certain level of measured

confidence was screened out using standard recognizers. Then, two methods based on accent-adaptive models and features respectively were developed, and the speech recognition with and without accent adaptation were compared. The new accent-adaptive models were shown to improve the performance of the standard algorithms currently used in industry.

1.3 Overview of the Thesis

In the next chapter, we review the current literature in this field and provide a summary of the state of the art. In Chapter 3 and 4, two driving areas of application are discussed in detail. The first is language learning, the topic of Chapter 3. The second is speech recognition adaptation, the topic of Chapter 4. In both chapters, we will discuss the problem and within the approach we plan to explore to advance the current state of the art. Finally in Chapter 5, conclusions are presented. Some of the work published in [6–9] is included in this thesis, with permission of the publishers.

2. BACKGROUND AND LITERATURE REVIEW

2.1 Methods for Mispronunciation Detection

In this section, the leading methods of mispronunciation detection are reviewed: (i) Likelihood Ratio Test (LRT) based methods, (ii) posterior probability based methods and (iii) predictive feature based methods.

2.1.1 Likelihood ratio test based methods

In mispronunciation detection, the goal is to determine if the pronunciations of phones are correct. The Likelihood Ratio Test (LRT) is a well established to tackle verification problems, such as mispronunciation detection.

Franco et al. [2] first introduced the LRT method to detect mispronunciations. In their approach, two models representing correct (λ_C) and mispronounced (λ_M) pronunciations are trained separately. Given an observation sequence $\boldsymbol{o} = \boldsymbol{o}_{t_0}\boldsymbol{o}_{t_0+1}$ ... $\boldsymbol{o}_{t_0+d-1}$ in an utterance s, the log-likelihood ratio score $\mathrm{LLR}(\boldsymbol{o}, q)$ can be computed as

$$\mathrm{LLR}(\mathbf{o}, q) = \frac{1}{d} \sum_{t=t_0}^{t=t_0+d-1} [\log p(\boldsymbol{o}_t|q, \lambda_M) - \log p(\boldsymbol{o}_t|q, \lambda_C)]. \tag{2.1}$$

In this equation, t_0 is the initial frame index of phone class q in s, d is the frame duration of the observation $\boldsymbol{o}$ in q, and $\boldsymbol{o}_t$ is the acoustic observation for the tth frame of s. The score $\mathrm{LLR}(\boldsymbol{o}, q)$ is compared with a phone-dependent threshold to determine if the pronunciation of q is correct or not.

Ito et al. [10] also introduced a similar method to detect mispronunciations. They constructed a correct pronunciation model α for a word or an utterance by concatenating the Hidden Markov Models (HMMs) of each phone. Then, they replaced

some phones with some potential variations using the speakers' mother tongue and generated the mispronunciation models $\bar{\alpha}_1, \bar{\alpha}_2, \cdots, \bar{\alpha}_K$. These models represent different types of mispronunciations in the phone sequence. The log-likelihood difference $D(\boldsymbol{o}|\alpha, \bar{\alpha}_i)$ is then used to measure the level of mispronunciation and is computed as:

$$D(\boldsymbol{o}|\alpha, \bar{\alpha}_i) = L(\boldsymbol{o}|\alpha) - L(\boldsymbol{o}|\bar{\alpha}_i). \tag{2.2}$$

The authors significantly improved the performance by choosing multiple thresholds for different mispronunciation classes. The thresholds are determined by using a decision tree, formulated as

$$D(\boldsymbol{o}|\alpha, \bar{\alpha}_i) = L(\boldsymbol{o}|\alpha) - L(\boldsymbol{o}|\bar{\alpha}_i) - \theta(R(\bar{\alpha}_i)), \tag{2.3}$$

where $R(\bar{\alpha}_i)$ and $\theta(R(\bar{\alpha}_i)$ represent the rule to generate mispronounced model $\bar{\alpha}_i$ and its corresponding threshold. The tree-based clustering method used to determine the class is explained well in [10].

These methods generally require building a model for the real mispronunciation distribution to cover all types of mispronunciation variation. However, since the distribution of mispronunciations differ with different speakers and texts, it is difficult to obtain this mispronunciation model when the training corpus with human transcription is limited.

2.1.2 Posterior probability based methods

Franco et al. [2] also proposed another method to measure the mispronunciations based on posterior probability. Given observation sequence $\boldsymbol{o}$ with phone class q. this method can be formulated below:

$$P(q|\boldsymbol{o}) = \frac{1}{d} \sum_{t=t_0}^{t=t_0+d-1} \log \left(\frac{p(\boldsymbol{o}_t|q)P(q)}{\sum_{i=1}^{Q} p(\boldsymbol{o}_t|q_i)P(q_i)} \right) \tag{2.4}$$

where t_0, d and $\boldsymbol{o}_t$ have the same definitions as in Eq. (2.1), $P(q)$ is the prior probability of the phone class q, and Q is the number of the labels.

Witt et al. [11] detected mispronunciations using a method called Goodness of Pronunciation (GOP), which in fact can be derived from the original posterior method proposed by Franco et al. Given observation sequence $\boldsymbol{o}$ and its label q, this method can be formulated as

$$\begin{aligned} \mathrm{GOP}(\boldsymbol{o}, q) =& |\log(p(q|\boldsymbol{o}))|/d \\ =& \left|\log\left(\frac{p(\boldsymbol{o}|q)P(q)}{\sum_{i=1}^{Q} p(\boldsymbol{o}|q_i)P(q_i)}\right)\right| \Big/ d. \end{aligned} \tag{2.5}$$

Assuming uniform distribution for all phones and approximating the summation by taking the maximum value, this equation can be simplified to

$$\mathrm{GOP}(\boldsymbol{o}, q) = \left|\log\left(\frac{p(\boldsymbol{o}|q)}{\max_{1\leq i\leq Q} p(\boldsymbol{o}|q_i)}\right)\right| \Big/ d. \tag{2.6}$$

Forced alignment is used to compute the numerator (2.6) with the transcription. The denominator is computed by recognition using a free running phone loop grammar in their method [11].

These types of methods require the approximation of the prior distribution of observation $P(\boldsymbol{o})$. In Franco et al.'s work [2], the approximation of $P(\boldsymbol{o})$ is computed at the frame level, while in Witt et al.'s work [11], it is computed using a free running phone loop. However, this approximation requires the posterior distributions of each phone class q, which is difficult to obtain. Thus, computing $p(q|\boldsymbol{o})$ is more time-consuming in Posterior Probability based methods than in LRT methods and needs more native training samples.

2.1.3 Predictive feature based methods

Methods based on predictive features focus on selecting effective features to distinguish recognized results from other possible errors correctly. The classifier can be based on pattern recognition techniques, examples of which include decision tree, artificial neural nets, etc. Recently a couple of other classifiers have received attention like Support Vector Machine (SVM) and Boosting. According to Jiang's review [12], the predictive features may have to be obtained as part of the detection process at levels of acoustics, language models, semantics and syntax. Common predictive features reported in the literature include: 1) pure normalized likelihood score, 2) N-best list score, 3) acoustic stability, 4) hypothesis density, 5) duration, 6) language model (LM), 7) Parsing, 8) Posterior probability, and 9) log likelihood ratio.

For example, if the log likelihood ratios are selected for native and non-native classification, for each training samples of the phone segment to be pronounced, the feature vector will be

$$\boldsymbol{f} = [\mathrm{LLR}(\boldsymbol{o}|q, q_1), \mathrm{LLR}(\boldsymbol{o}|q, q_2), \cdots, \mathrm{LLR}(\boldsymbol{o}|q, q_Q)]^T \tag{2.7}$$

where $\boldsymbol{o}$ is the acoustic realization of q (the canonical label that is pronounced); $q_1, q_2, \cdots, q_Q$ depicts the whole set of phone classes. $\mathrm{LLR}(\boldsymbol{o}|q, q_i)$ denotes the ith log likelihood ratio defined as

$$\mathrm{LLR}(\boldsymbol{o}|q, q_i) = \log p(\boldsymbol{o}|q_i) - \log p(\boldsymbol{o}|q). \tag{2.8}$$

If there are R training samples of the phone segment q with pronunciation labels $t_1, t_2, \cdots, t_R$ to indicate the class of mispronunciation, either "correct" or "mispronounced", these features can be used to train the binary classifier using SVM [13, 14]

or Boosting [15] and can form the discriminative function for the acoustic observation $\boldsymbol{o}$ and canonical label of phone class q as,

$$d(\boldsymbol{f}) = \boldsymbol{w}^T \boldsymbol{f}, \tag{2.9}$$

where $\boldsymbol{w}$ is the vector of weights obtained for each dimension in the feature vector. The output value of this function will be compared with threshold θ_q to determine whether the testing phone segment is correct or mispronounced.

2.2 Performance Results

Results of mispronunciation detection or pronunciation scoring can be obtained at the phone level, word level, phrase level, sentence level or speaker level. For language learning, each level from smallest to largest can be relevant. However, for speech recognition adaptation, the main task is to identify the speaker's accent, so the mispronunciation detection usually focuses on the profile level to detect the speaker's mother tongue, namely classify the speaker into one of the accent categories.

For language learning, Franco et al. [16] developed a toolkit for speech recognition and pronunciation scoring called EduSpeak® on Spanish in 2010. The native corpus consists of 26,571 newspaper sentences from 142 native Latin American Spanish speakers. The non-native corpus contained 14,000 newspaper sentences, read by 206 adult American English speakers each of whom had some knowledge with Spanish. The word recognition error Franco et al measured before and after speech recognition adaptation is 6.9% and 3.1% [16]. Mispronunciation detections up to the sentence level are measured in various ways, results of which are listed in Table 2.1. The human-machine correlations for non-native Spanish speakers across the full spectrum of measures range from 0.414 to 0.623.

For determining the degree of accent of speakers, in 2007, Bartkova and Jouvet [17] collected a speech corpus from 79 French, 171 English, 200 German and 81 Spanish speakers pronouncing 83 French words and expressions through the telephone. All

Table 2.1 Human-machine correlations up to sentence level with mapped and combined machine score

Machine scores	correlation for native/non-native Spanish speakers	correlation for non-native Spanish speakers only
Posterior	0.642	0.585
Phone duration	0.548	0.414
Word duration	0.606	0.449
Phone duration + word duration	0.611	0.455
Posterior + phone duration	0.680	0.594
Posterior + word duration	0.709	0.611
Posterior + phone duration + speech rate	0.780	0.607
Posterior + word duration + speech rate	0.785	0.623

samples were then labeled with degree of accent from 0 (no accent) to 7 (very heavy accent) by an expert phonetician. First, they performed text-dependent mispronunciation detection with both "native" (French speakers only) and "accent-adapted" (all speakers) models based on the conventional MFCC-HMM framework and showed the performance degradation in the non-native with "native" model and in the native with "accent-adapted" model. Then, they constructed a performance basis by comparing the human subjective score of accent level with ASR results based on the "native" model. The manual accent annotation and automatic scoring performance is illustrated in Table 2.2. Finally, they developed an accent-adapted phone-level model using free phoneme loop for each type of language, and performed mispronunciation detection to determine the degree of accents given the speaker's language origin. They claimed to reduce the error rate by 18% to 42% from light to strong accent degree.

Since the non-native speech databases for mispronunciation detection [18] are relatively limited compared with standard databases for ASR, current and proposed work in this thesis will be based on a native/non-native Spanish isolated word corpus provided by the department of Foreign Languages and Literatures (FLL) at Purdue West Lafayette and customer support call data in English from Interactive Intelligence

Table 2.2 Manual accent annotation and automatic recognition performance

Expert notation	Degree of accent	Number of speakers				Error rate (%)
		FR	EN	DE	SP	
no-accent	0	110	14	57	11	6.7
very slight	1	2	11	26	4	6.3
slight	2	1	23	73	13	6.7
slight mid	3		13	16	7	8.2
mid	4		33	28	16	11.2
heavy mid	5		10		6	18.9
heavy	6	1	52	3	15	30.2
very heavy	7		25		8	38.2

FR: French; EN: English; DE: Deutsch (German); SP: Spanish

Inc. (I3). Publicly available and commonly used corpuses with non-native data, like TIMIT, will also be considered later on.

3. MISPRONUNCIATION DETECTION FOR LANGUAGE LEARNING

3.1 Overview

As discussed in Section 1.1, phone-level mispronunciation detection systems for language learning usually detect mispronunciations in units of phones, words, phrases, or sentences. To perform mispronunciation detection, the speaker's speech samples are first converted to certain types of features, such as Mel Frequency Cesptral Coefficients (MFCC). Then a classifier is applied using statistical models, such as HMMs and SVMs. This allows the system to determine if the input has been mispronounced or to classify the mispronunciation by type.

The conversion of the cepstral coefficients from the linear scale to Mel frequency scale is an important step used to generate the MFCCs. In turn, it allows the system to mimic the audio discrimination processes used by humans. This frequency warping process is illustrated in Figure 3.1. As part of this thesis, we introduce a new optimized warping function customized at the syllable and phonetic levels. This customized warping function leads to Adaptive-frequency Cepstral Coefficients (AFCCs) that we show can result in improved performance. More specifically, AFCCs can be used to emphasize the discriminative information of the particular phone and hence help to distinguish that particular phone out of the rest of phones, including the accented variation of that phone.

If enough native pronunciations for each phone $p_1, p_2, \cdots, p_Q$ in the whole phonetic alphabet [19] of the targeted language are obtained by segmenting phones from the native speech corpus, the frequency warping scale for each phone can be optimized to separate the phone detection scores of one particular phone q_i from all the other phones p_j, $j \in (1, 2, \cdots, Q), j \neq i$ in the whole phonetic alphabet. Or if enough non-

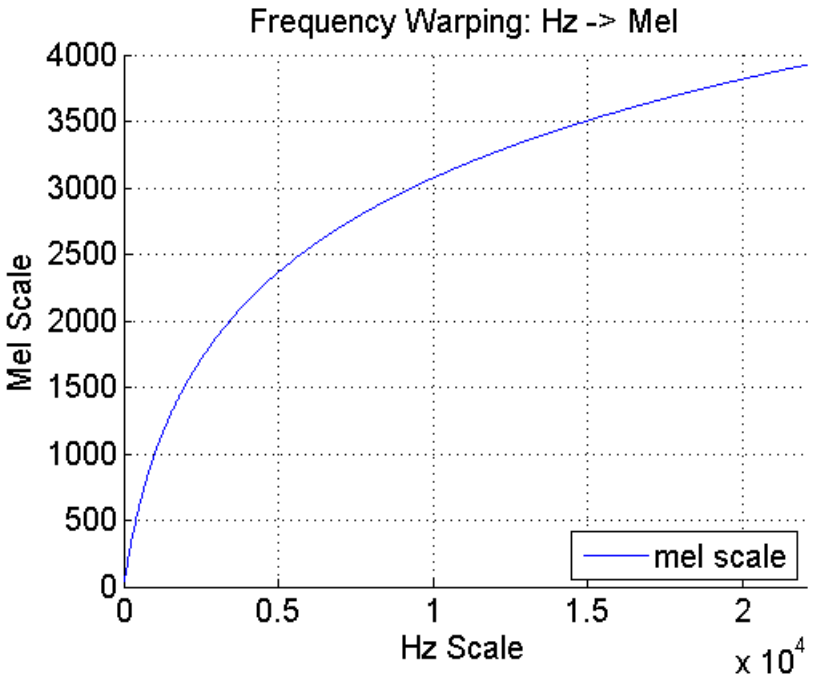


Fig. 3.1. Frequency warping from linear scale (Hz) to Mel scale (Mel)

native pronunciations are also available for segmenting accented phones covering the phonetic alphabet, the frequency scale for each phone can be optimized to match human subjective scores of both native and non-native pronunciations (maximizing the correlation between human and computer scores of phones). The second optimization criteria should directly lead to higher accuracy of mispronunciation detection. However, it also requires a non-native corpus large enough to cover all the possibilities of mispronunciations and avoid dependency on a limited non-native corpus.

3.2 Baseline System

The baseline phone-level mispronunciation detection system for language learning used in this thesis is based on a MFCC-HMM framework. The utterance to be detected can be either word, phrase or sentence. The system will detect mispronunciations at the phone level and indicate the mispronounced phones in the utterance. It uses MFCCs as the feature input and uses HMM-based statistical models to perform

mispronunciation detection. Developing this system involves training, validation and testing. These processes are explained below and focus on Spanish word mispronunciation detection.

3.2.1 System Design Based on HMM

During system training, for each word $W^{(l)}$, where $n \in (1, 2, \cdots, L)$ and L is the size of the word vocabulary, a portion of the native pronunciation samples are selected to train the HMM parameters $\lambda = (A, B, \pi)$, where

$$\lambda = \begin{cases} A = \{a_{ij}\} = P(q_t = j | q_{t-1} = i) & i, j \in [1, N], t \in [2, T] \\ B = \{b_j(\boldsymbol{o}_t)\} = P(\boldsymbol{o}_t) | q_t = j) & j \in [1, N], t \in [1, T] \\ \pi = \{\pi_i\} & i \in [1, N] \end{cases} \tag{3.1}$$

for that particular word using the Forward-Backward Algorithm [20].

A is the matrix of state transition probabilities; B is the posterior observation probability given state j at observation feature $\boldsymbol{o}_t$ at time frame t; π_i is the initial probability of state i; N is the total number of states, which normally can be specified according to the number of phones in that word plus two states of silence at both ends; and T is the total number of time frames for pronunciation samples. After training, each training sample is segmented into phone sequences.

During system validation, a portion of the native and the non-native samples, all of which have certain levels of mispronunciation, will be selected to find the optimal threshold for each phone q_i, $i \in (1, 2, \cdots, Q^{(l)})$ of each word $W^{(l)}$, where $Q^{(l)}$ is the number of phones in word $W^{(l)}$. All these samples belong to two classes, either accented, or non-accented, and they can be segmented into phones by force alignment using the Viterbi Algorithm. The posterior probability $P(q_i|\boldsymbol{o})$ of each phone can be calculated by Equation (2.4) in Section 2.1.2. After obtaining these two groups of posterior probabilities for native and non-native samples of each phone in word $W^{(l)}$, they will be rescaled to normal distribution $N(\mu_i^{(l)}, \sigma_i^{(l)})$ and considered as computer

scores to compare with human subjective scores. The threshold $\theta_i^{(l)}$ at phone q_i of word $W^{(l)}$ is optimized at the Equal Error Rate (EER) as shown in Fig. 3.2.

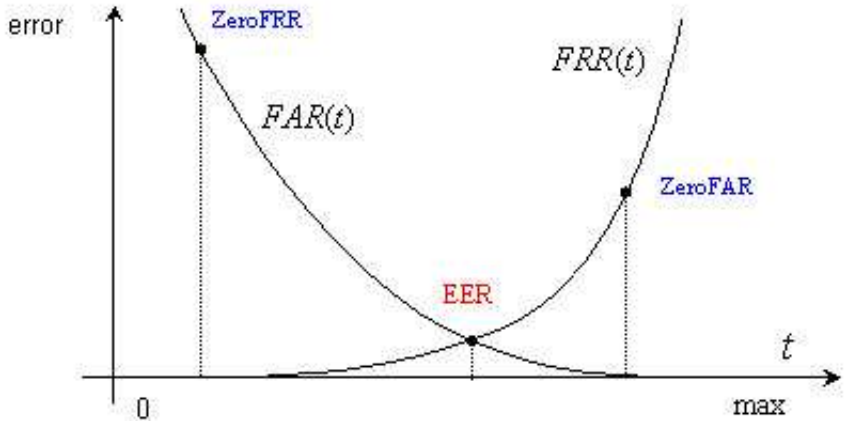


Fig. 3.2. EER determined by False Acceptance Rate (FAR) and False Rejection Rate (FRR)

During system testing, the rest of the native and the non-native samples will be used to test the performance of the mispronunciation detection. The simplified word mispronunciation detection system is illustrated in Fig. 3.3. Each phone $q_i^{(l)}$ of the pronounced word $W^{(l)}$ will be scored and an overall score of the word sample will be given by weighted average of each phone score

$$S_{\text{computer}}^{(l)} = \left[\sum_{i=1}^{Q^{(l)}} \left(d_i S_{i,\text{computer}}^{(l)}\right)\right] \Big/ \sum_{i=1}^{Q^{(l)}} d_i, \tag{3.2}$$

where $S_{i,\text{computer}}^{(l)}$ and d_i are the computer score and the frame length of phone q_i of word $W^{(l)}$ respectively. $Q^{(l)}$ is the number of phones for word $W^{(l)}$.

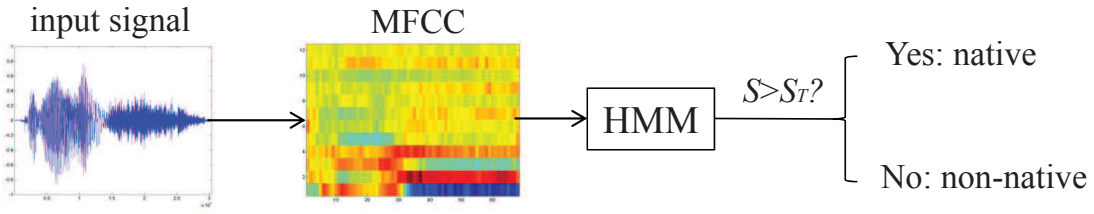


Fig. 3.3. Simplified word mispronunciation detection system

3.2.2 MFCC Extraction

Since the MFCCs are commonly used for speech recognition and mispronunciation detection, in this section, the procedure of extracting conventional MFCCs may be described as follows [21]:

1. Compute the Short Time Fourier Transform (STFT) of a windowed segment of the speech;
2. Take the magnitude squared to obtain the power spectrum;
3. Warp the power spectrum to the Mel scale, via overlapping triangular windows;
4. Compute the log of the power spectrum;
5. Then take the Discrete Cosine Transform (DCT) to obtain a ceptral domain representation;
6. The MFCCs are the amplitudes of the resulting spectrum.

The above procedure is illustrated in Fig. 3.4 and can be described mathematically as

$$\text{MFCC} = \text{DCT}(\log\{mel[\text{STFT}(s)]^2\}), \quad s \text{ is the windowed signal} \tag{3.3}$$

A comparison of linear warping and Mel warping from the power spectrogram to the audio spectrogram is given in Fig. 3.5. Given the power spectrogram on the left, if linear warping if performed, the frequency components are equally spaced in the audio spectrogram and the only difference from the power spectrogram is caused by the windowing effect, while in the figure on the bottom right, the frequency components expand and shrink in the lower and higher portions respectively.

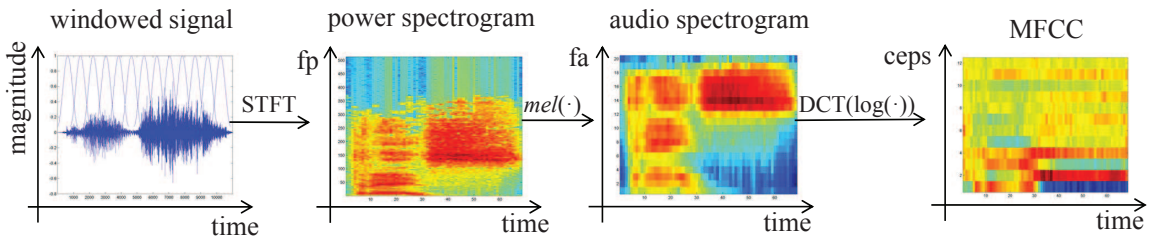


Fig. 3.4. Procedure to generate MFCC

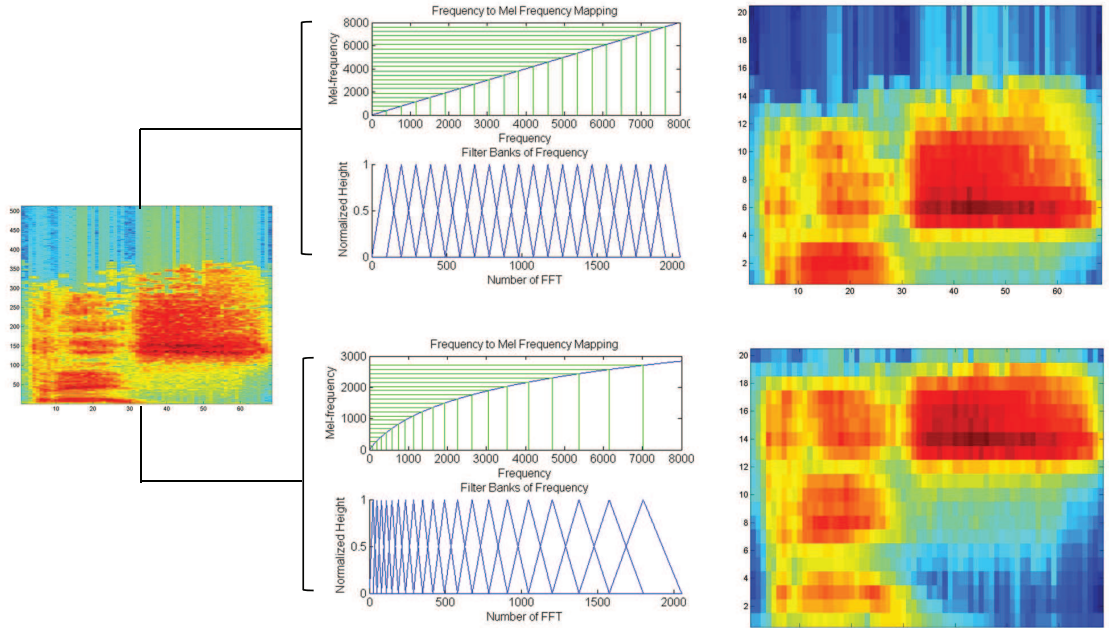


Fig. 3.5. Comparison of a) linear and b) Mel scale warping

3.3 Improvement Based on Adaptive Features

As discussed in Section 3.1, the text-adaptive (or phone-adaptive) features, can be obtained in a couple of ways: a) by maximizing the separation of target phones and all other phones including the variations of mispronounced target phones using the native corpus only; or b) by maximizing the correlation between the computer scores and human scores of phones. Both methods require a large native corpus as well as a large non-native corpus covering most of the variations of each phone. The second approach also requires human scoring.

The database provided by the Department of Foreign Language Learning (FLL) at Purdue contains 10 Spanish isolated words, each of which is pronounced 10 times by 20 males and 20 females. Male and female speakers are evenly selected from native Spanish and non-native English speakers learning Spanish. This database is not large enough to cover all phones in the Spanish phonetic alphabet and all variations of mispronunciation for each phone. Hence, two compromises were made for developing the text-adaptive features and testing the performance improvement:

- develop AFCCs based for each word instead of each phone
- adjust optimization criteria from "maximizing separation between target phone and all other phones" to "maximizing separation between accented and non-accented pronunciations of target words"

After selecting accented pronunciation samples from the non-native corpus, the two groups of HMM-based computer scores for non-accented and accented pronunciations are assumed to be well separated. The separation in terms of classification rate r can be measured using Bayesian Minimum Error (BME) by assuming these two groups of scores X_1 and X_2 are both under normal distribution with $X_1 \sim N(\mu_1, \sigma_1^2), X_2 \sim N(\mu_2, \sigma_2^2)$ and $P(\omega_1) = P(\omega_2) = 1/2$ where ω_1 and ω_1 denote the classification of two groups, $P(\text{error})$ denotes the Bayesian Minimum Error Rate, which can be calculated from the intersection area of the two distributions shown graphically in Fig. 3.6. $P(\text{correct})$ can be calculated as

$$\begin{aligned} P(\text{correct}) &= 1 - (\int_{x^*}^{\infty} p(x \mid \omega_1) P(\omega_1) \mathrm{d}x \\ &\quad + \int_{-\infty}^{x^*} p(x \mid \omega_2) P(\omega_2) \mathrm{d}x), \end{aligned} \tag{3.4}$$

where x^* can be found by computing the discriminant function $g(x)$ at $g(x) = 0$ and

$$g(x) = p(x \mid \omega_1) P(\omega_1) - p(x \mid \omega_2) P(\omega_2). \tag{3.5}$$

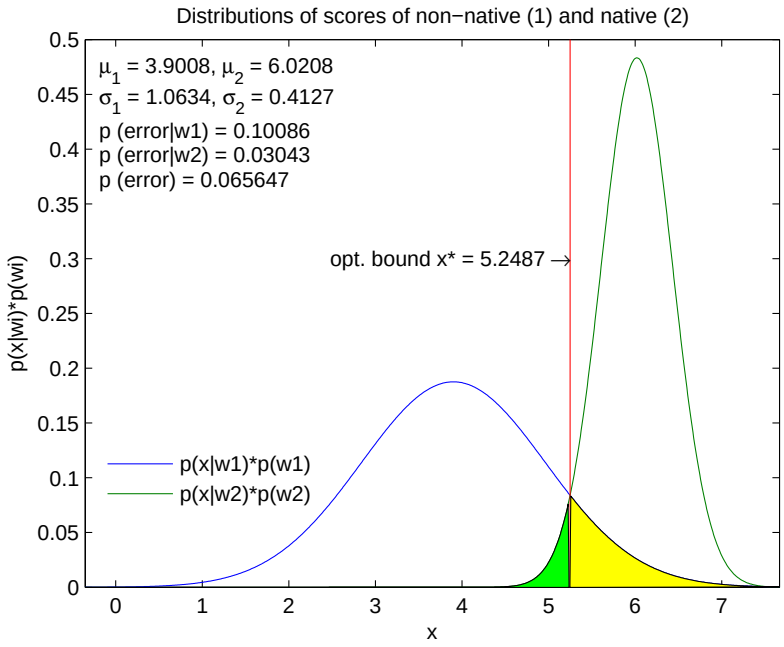


Fig. 3.6. Distributions of HMM scores of two groups with correct and incorrect pronunciations.

3.3.1 Adaptive Frequency Scale

Frequency Scale Generation

To develop the algorithm, a framework is needed in which the frequency scales can be generated. Recognizing that the scale is bounded between 0 and π and that we can arguably expect the optimal scale to be one that concentrates its sensitivity in the low frequency region, we first considered a simple μ-law construction. This approach has the advantage that it can approximate the warping profile we anticipate and it only involves a single parameter. To obtain higher accuracy, however, we have based our algorithm on a dual parameter model (i.e. involving two degrees of freedom) where Piecewise Cubic Hermite Polynomial (PCHP) interpolation [22] is used to create the adaptive warping function.

For comparison, we use a) the HTK Mel scale (used in the HTK Speech Recognition Toolkit) [23], b) the Mel scale reported by Slaney [24] and c) the Bark scale

reported by Zwicker [25]. Mel and HTK Mel scales essentially employ the same equation and triangular filter banks but differ in their normalization. More specifically, the HTK Mel scale filter banks are normalized to have the same height while the other is normalized to have constant passband area. Fig. 3.7 shows HTK Mel, Mel and Bark frequency scales compared with an example of the new PCHP interpolated scale. For convenient display, all scales are normalized to the Nyquist frequency F_N ($F_N = F_s/2 = 22050$ Hz in this project. F_s is the sampling frequency). The PCHP interpolation provides smooth interpolation and a monotonically increasing frequency scale similar to the Mel-scales, but with much more flexibility since the interpolation point used to define the curve can be placed anywhere in the bi-frequency plane. This point is also illustrated in Fig. 3.7.

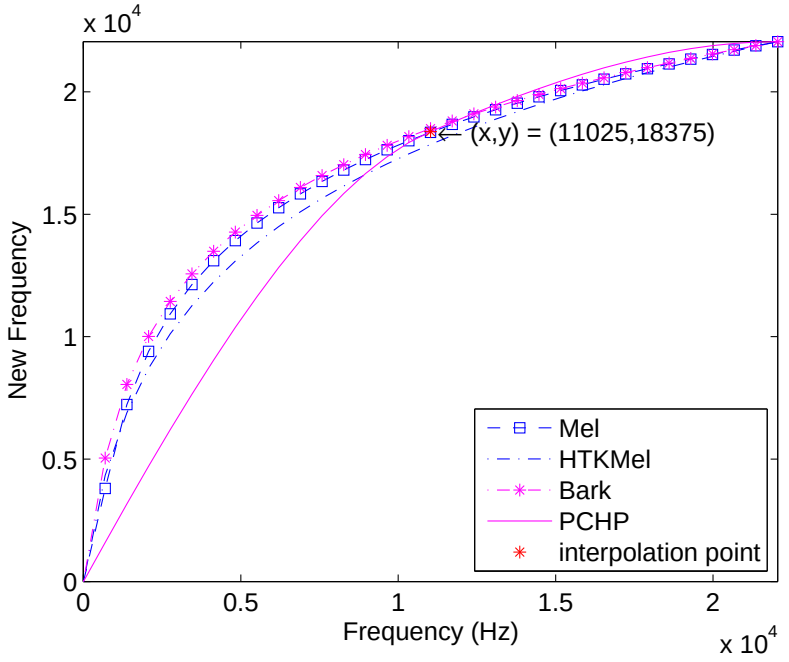


Fig. 3.7. Comparison of multiple frequency scales.

Adaptive Frequency Scale Optimization

The procedure for PCHP frequency scale optimization can be described as follows. Given any interpolation point (x_i, y_i) in the bi-frequency plane, first, compute the PCHP interpolated frequency scale. Based on that scale, compute the AFCCs for all samples from both native and non-native groups. Using AFCC(x_i, y_i) as inputs to the Leave-One-Out HMM training and testing (discussed in Section 3.3.2), all samples are assigned HMM scores $S_{\text{HMM}}(x_i, y_i)$. These HMM scores are then rescaled to the range 1-7, which serves as a measurement of pronunciation quality. The classification rate $r(x_i, y_i)$ is computed based on the distribution of the rescaled scores $S_{\text{Rescaled}}(x_i, y_i)$ of these two groups using the Bayesian rule.

Thus, through the process $(x_i, y_i) \rightarrow \text{AFCC}(x_i, y_i) \rightarrow S_{\text{HMM}}(x_i, y_i) \rightarrow S_{\text{Rescaled}}(x_i, y_i)$ $\rightarrow r(x_i, y_i)$, any point (x_i, y_i) on the bi-frequency plane can be eventually mapped to a certain classification rate $r(x_i, y_i)$. The frequency scale optimization is an iterative procedure consisting of finding (x^*, y^*), where

$$(x^*, y^*) = \underset{x,y \in [0, F_s/2]}{\text{argmax}} \left(r(x, y)\right) \tag{3.6}$$

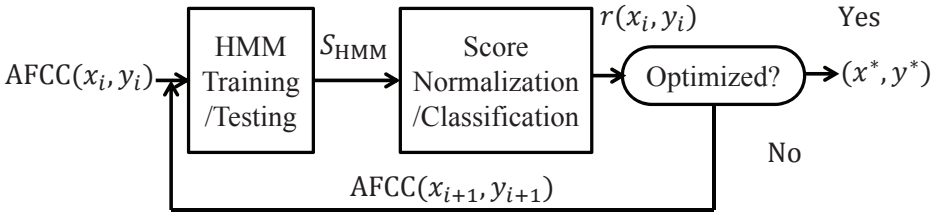
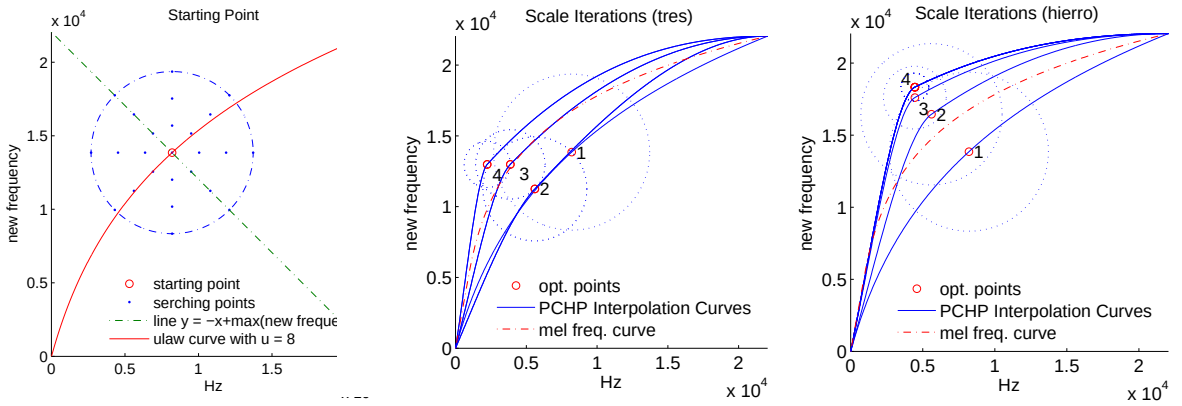


Fig. 3.8. Procedure of frequency scale optimization

The above procedure is depicted in Fig. 3.8. In order to find (x^*, y^*) that maximizes $r(x, y)$, a modified N-step search algorithm similar to the 3-step search algorithm in motion estimation is employed:

1. Initialization: a) choose a starting interpolation point $(x_0, y_0)^{(1)}$ in the bi-frequency plane. A good rule of thumb is to use the "center" of the μ-law scale at $\mu = 8$ (the intersection of μ-law scale and the diagonal between the maximum



(a) (b) (c)

Fig. 3.9. Illustration of the PCHP optimization algorithm. (a) Starting point and search points for optimization; (b) illustration of four iterations of the algorithm for the word `tres` and (c) four iterations of the algorithm for the word `hierro`.

frequencies in both normal and new frequency axes); b) set the search region as a circle centered at $(x_0, y_0)^{(1)}$ with radius $R(1) = \frac{\pi}{2}$ and c) select M candidate searching points $(x_i, y_i)^{(1)}, i = 1, 2, ..., M$ ($M = 24$ in this project) that are evenly spaced on concentric circles within the search region as illustrated in Fig. 3.9(a).

2. Iteration: in each iteration n, compute the corresponding classification rate $r(x_i, y_i)^{(n)}$ at each candidate point $(x_i, y_i)^{(n)}$ and set the current optimal classification rate $r(x^*, y^*)^{(n)} = \max[(r(x_i, y_i)^{(n)}]$.

- If the current optimal point $(x^*, y^*)^n$ is optimal through all iterations up to n, i.e. $(x^*, y^*)^n = \text{argmax}(r(x, y))^{(j)}, j = 1, 2, ..., n)$, then, set a) the next starting point $(x_0, y_0)^{(n+1)} = (x^*, y^*)^{(n)}$ and b) the next search region at the circle centered at $(x_0, y_0)^{(n+1)}$ with radius $R(n+1) = \max[\frac{\pi}{2^{n+1}}, |((x_0, y_0)^{(n+1)} - (x_0, y_0)^{(n)}|]$; c) similarly select M candidates $(x_i, y_i)^{(n+1)}, i = 1, 2, ..., M$, evenly spaced within the new search region;

- Else, set a counter $k = k + 1$ ($k = 0$ initially); keep the starting point, the search region and the search candidates the same in the next iteration, i.e. $(x_0, y_0)^{(n+1)} = (x_0, y_0)^{(n)}$, $R(n+1) = R(n)$ and $(x_i, y_i)^{(n+1)} = (x_i, y_i)^{(n)}$. This repetition compensates for the small variation of HMM scores due to the randomization in HMM training with a relatively small database.

3. Termination: if the counter $k = K$, where $K = 3$ in this project, then terminate the iteration. The point providing the largest classification rate is $(x^*, y^*) = \text{argmax}(r(x^*, y^*)^{(j)}, j = 1, 2, ..., n)$.

Finally, since the radius of the search region R is converging in each iteration, we are able to find the locally optimal word adaptive scale (subject to the constraints of the construction) provided by the largest $r(x^*, y^*)$.

3.3.2 Experiments and Results

Database and Human Scoring

In this mispronunciation detection project, the talkers are 20 native speakers of English who have completed one year of instruction in college-level introductory Spanish and 20 university students whose native language is Spanish. 10 Spanish words comprise the corpus. Each speaker pronounces each word 10 times. The human scoring juries are composed of 22 adult native speakers of Spanish. Scores range from 1 (poor) to 7 (excellent) based on the level of mispronunciation. In the training of correct and incorrect pronunciation groups, outlying samples are removed so that the samples within each group are more homogeneous. Figure 3.10 shows the GUI of the Spanish word scoring system used in this work.

Optimization System Setup

All samples are noise suppressed and times-scale modified. MFCCs and AFCCs used in this study are 13 dimensional with twelve cepstral coefficients and one energy coefficient.

The HMMs are trained and tested five times using the Leave-One-Out (LOO) algorithm. During each trial, 80% of the native samples are trained and the remaining 20% native and all the non-native samples are tested so that every sample has a score (or averaged score).

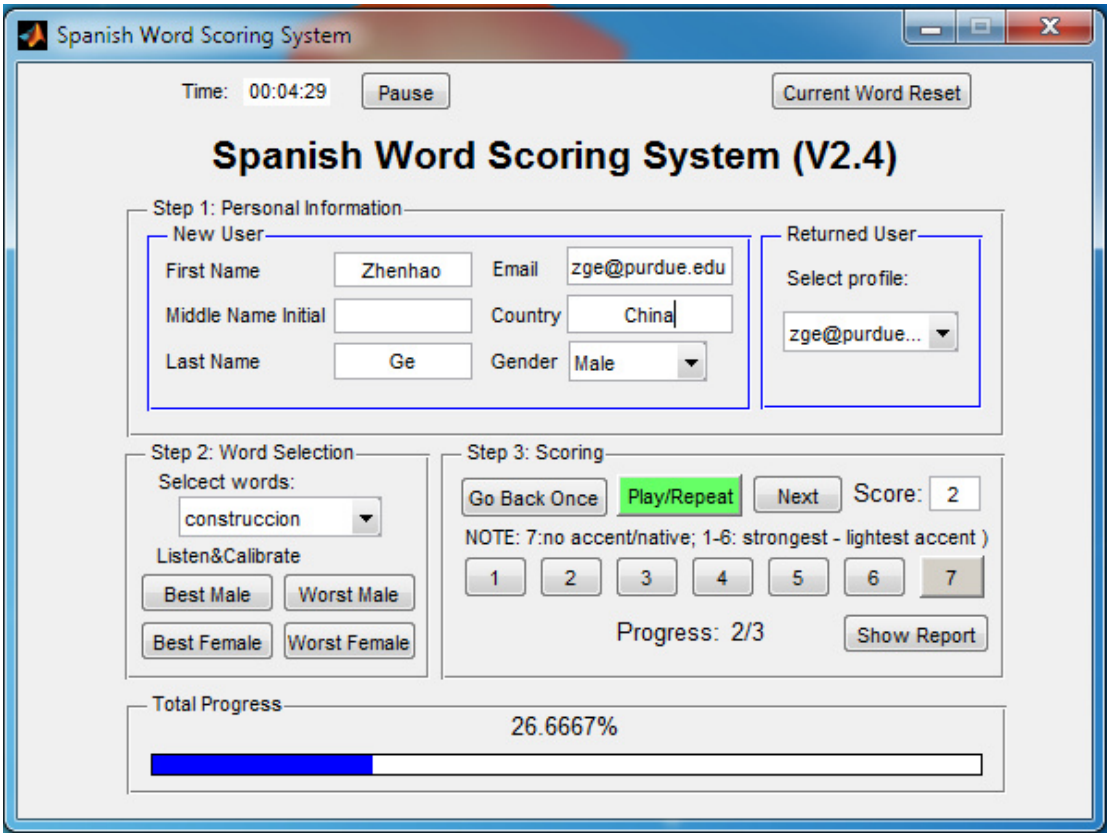
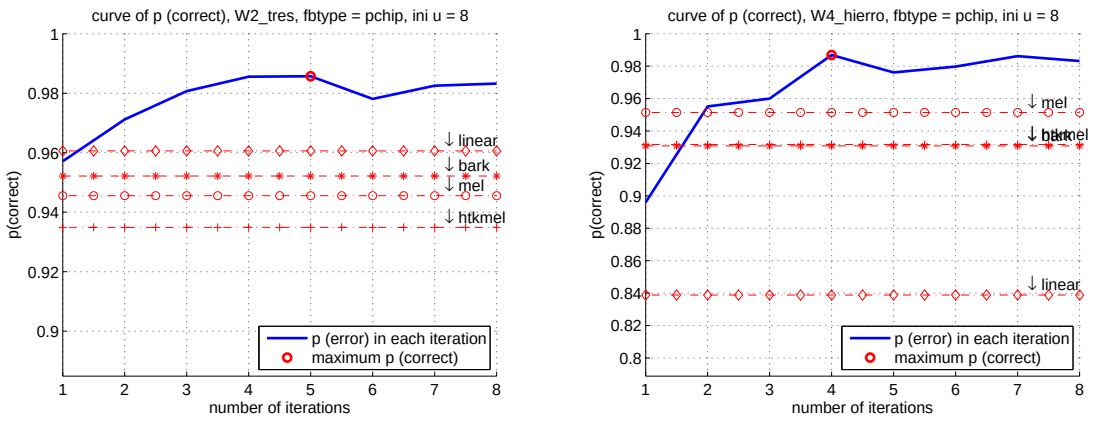


Fig. 3.10. Spanish word scoring system used for human scoring

Experimental Results

Fig. 3.11 shows the evolution of classification rate $r(x^*, y^*)^{(n)}$ compared with the classification rates associated with Mel, HTK Mel, Bark, and linear scales. The classification rate for the adaptive frequency scale converges very fast and the optimized AFCCs outperform the conventional MFCC systems. The variations that can be seen after the second iteration are due to randomization in the HMM training.

The AFCC result in better frequency scales and better separation of the two groups in terms of classification rate. The improvement is illustrated in Table 3.1, which shows a comparison of multiple frequency scale performances for 10 Spanish words.



(a) (b)

Fig. 3.11. Classification rates for successive AFCC iterations contrasted against the corresponding rates for the Mel scale and linear scale. The test words used here are `tres` (a) and `hierro` (b).

Table 3.1 Classification rates for different frequency scales

No.	Word	Adaptive	Mel	HTKMel	Bark	Linear
1	`jamaica`	96.97%	93.34%	93.32%	96.52%	92.29%
2	`tres`	98.57%	94.55%	93.49%	95.21%	96.06%
3	`gemelas`	97.23%	95.29%	97.02%	94.80%	93.24%
4	`hierro`	98.69%	95.15%	93.17%	93.08%	83.88%
5	`pala`	93.12%	82.61%	84.40%	84.40%	84.47%
6	`tordurados`	97.76%	90.29%	94.98%	96.45%	83.52%
7	`accidente`	98.26%	93.20%	93.66%	94.35%	89.27%
8	`construccion`	97.15%	96.63%	93.12%	95.70%	91.26%
9	`puertorriquena`	97.89%	95.74%	95.38%	94.18%	84.89%
10	`aire`	96.17%	92.82%	93.21%	93.68%	89.31%
	Average	97.18%	92.82%	93.21%	93.68%	89.31%

3.4 Alternative Methods Using PCA for Small Database

In this section, a new mispronunciation detection and classification algorithm based on Principle Component Analysis (PCA) is discussed. It is hierarchical with each successive step refining the estimate to classify the test word as being either

mispronounced or correct. Preprocessing before detection, like normalization and time-scale modification, is implemented to guarantee uniformity of the feature vectors input to the detection system. The performance using various features including spectrograms and Mel-Frequency Cepstral Coefficients (MFCCs) are compared and evaluated. Best results were obtained using MFCCs, achieving up to 99% accuracy in word verification and 93% in native/non-native classification. Compared with Hidden Markov Models (HMMs) which are used pervasively in recognition application, this particular approach is computational efficient and effective when training data are limited. The approach used in this section for detecting mispronunciations is to classify pronunciations at both the word-level and syllable-level as either acceptable or unacceptable.

3.4.1 PCA Method for Patten Recognition

PCA is a well-known method that has been successfully employed in high-dimensional recognition problems. One of the popular applications of PCA is for face recognition, which has been studied extensively over the past 20 years [26–30]. The fundamental idea behind PCA is encoding the most relevant information that distinguishes one pattern from another and eliminating dimensions with less information to reduce the computational cost [31].

The mispronunciation detection work in this paper shares much in common with face recognition. Mispronunciation detection can be thought of as a pattern recognition problem, just like face recognition. Furthermore, Mel-Frequency Cepstral Coefficients (MFCCs) and spectrograms, which are the features that we are investigating, are also of high dimension. The similarities between mispronunciation detection and face recognition motivated our incorporating PCA-based face recognition techniques into our work. In the next section, we first review the application of PCA in face recognition, and then discuss the procedures of the PCA-based mispronunciation detection.

Review of Face Recognition using PCA

The PCA approach for face recognition requires all data (face images), training data $\mathcal{D}_{\text{train}}$, and testing data $\mathcal{D}_{\text{test}}$ from multiple classes to have the same size (say $N_1 \times N_2$). The method tends to work best when the faces in all the images are located at the same position within the image. These data matrices are then vectorized to N_1N_2-dimensional column vectors. When implementing a PCA-based detection/classification system, like face recognition, given M vectorized training faces $\Gamma_1, \Gamma_2, \ldots, \Gamma_M (M \ll N_1N_2)$ as the whole set of $\mathcal{D}_{\text{train}}$, let $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$ be the mean face and let $\Phi_i = \Gamma_i - \Psi$ be the mean-shifted faces which form $A = [\Phi_1, \Phi_2, ..., \Phi_M]$. $C = AA^T$ is the covariance matrix of A, and can supply up to M eigenvectors to represent the eigenspace of A. Each face in A is a linear combination of these eigenvectors. If selecting the most significant M' eigenvectors to form a sub-eigenspace of training data $U = [u_1, u_2, \ldots, u_{M'}]$ $(N_1N_2 \times M')$, the representation of Φ_i in the M'-dimensional eigenspace is $\Omega_i = U^T\Phi_i$ and the projection of Φ_i in the eigenspace is $\hat{\Phi}_i = U\Omega_i$. The dimension of data is reduced from N_1N_2 to M' while preserving the most relevant information to distinguish faces.

After the eigenspace U for the training data A is computed, the following two steps are used for face recognition [30]:

1. Face detection: compute the Euclidean distance from test image Φ_j to its projection $\hat{\Phi}_j$ onto the eigenspace U,

$$e_{\text{dfes}} = \|\Phi_j - \hat{\Phi}_j\| \ , \tag{3.7}$$

   where e_{dfes} is called the "distance from eigenspace". If $e_{\text{dfes}} < T_d$, where T_d is a detection threshold, the test image is verified to be a face and step 2 below is used for face classification.

2. Face classification: compute e_{dies}, the minimum Euclidean distance between Ω_j and Ω_k

$$e_{\text{dies}} = \min \|\Omega_j - \Omega_k\| \qquad k = 1, 2, \ldots, K \tag{3.8}$$

where Ω_j is the eigenspace representation of the test face Φ_j and Ω_k is the averaged eigenspace representation of faces in class k, $k = 1, 2, \ldots, K$, where K is the number of classes. The parameter e_{dies} is called the "distance within eigenspace". If $e_{\text{dies}} < T_c$, where T_c is the classification threshold, then the test image is classified as a face belonging to that class; otherwise, a new class of faces is claimed. Figure 3.12 illustrates the difference between e_{dfes} and e_{dies}.

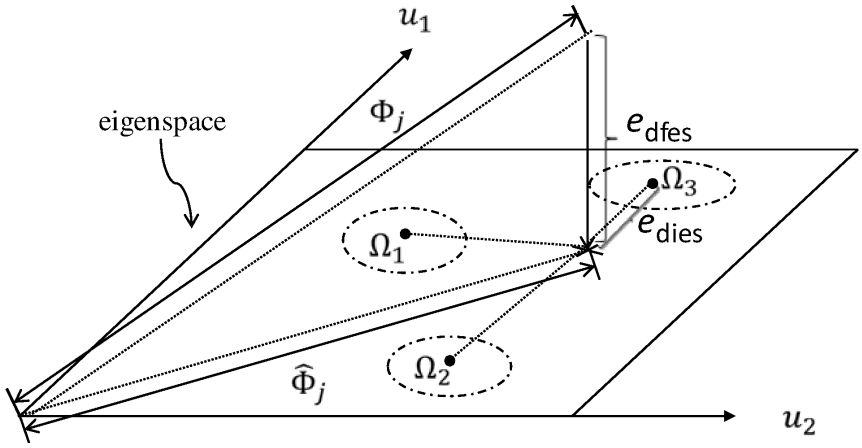


Fig. 3.12. A simplified version of an eigenspace to illustrate e_{dfes} and e_{dies}

Procedure for Mispronunciation Detection Using PCA

In this work, the training and testing data consist of two disjointed classes of speech samples. The first class consists of samples from native speakers, all of which have correct pronunciations. The second class contains samples from non-native speakers with possible mispronunciations. Given an input test sample, the procedure used for mispronunciation detection can be divided into three steps:

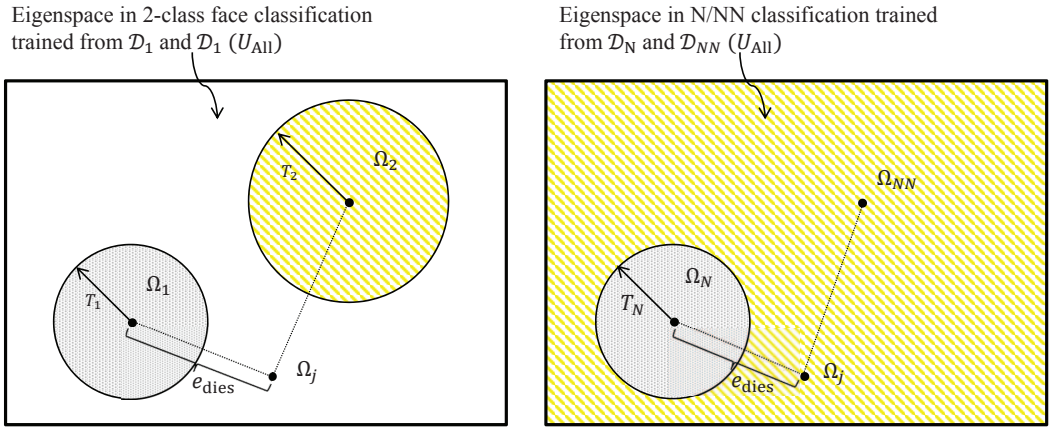
1. Word verification: compute e_{dfes} from the test sample Φ_j to the eigenspace U_{All} constructed from all samples in $\mathcal{D}_{\text{All.train}}$, where $\mathcal{D}_{\text{All.train}}$ includes all native $\mathcal{D}_{\text{N.train}}$ and non-native $\mathcal{D}_{\text{NN.train}}$ samples. If $e_{\text{dfes}} < T_d$, where T_d is the threshold for word verification, then the test sample is considered "verified" and we proceed to step 2. Otherwise the test sample is "rejected" and the detection process stops.

2. Native/non-native (N/NN) classification: compute e_{dfes} from the test sample Φ_j to the eigenspace U_{N} constructed only from $\mathcal{D}_{\text{N.train}}$. If $e_{\text{dfes}} < T_c$, then classify the test sample as native and stop. Otherwise, classify it as non-native and proceed to step 3.

3. Syllable-level mispronunciation detection: Divide the non-native test samples into syllables $\Phi_{jk}, k = 1, 2, \ldots, K$, where K is the number of syllables for that test sample. Similarly divide native samples $\mathcal{D}_{\text{N.train}}$ into syllables and call the k^{th} syllable $D_{\text{N}k\text{.train}}$. For each test syllable Φ_{jk}, compute its e_{dfes} to the eigenspace $U_{\text{N}k}$ constructed from k^{th} syllable of native samples $\mathcal{D}_{\text{N}k\text{.train}}$. If $e_{\text{dfes}} > T_k$, then classify the test syllable as mispronounced, otherwise classify it as correctly pronounced.

Word verification is very similar to face detection, since both cases use the distance metric e_{dfes} of the eigenspace U_{All} which is constructed from all classes of samples. However, N/NN classification is different from face classification. The main difference is that in face classification e_{dies} is used as the metric to determine the class of the "test" face, where as in mispronunciation detection we use e_{dfes} as the metric to do the N/NN classification. In face classfication, the various classes of faces are well defined. If e_{dies} of a "test face" lies within the predefined threshold of some class k, then it is classified as face k. Otherwise, it is claimed to be a "new face" that does not match any of the faces. The "new face" scenario for face classification is shown in Figure 4.5(a), where Ω_j is the "test" face. However, in N/NN classification, there are only two classes and any region outside the native class belongs to the non-native class.

So the misclassification shown in Figure 4.5(b) will occur if e_{dies} is used to determine the class of Ω_j. In Figure 4.5(b), Ω_j is in the non-native class but it is closer to the native class. Thus, instead of using distance e_{dies} within the eigenspace U_{All}, N/NN classification uses e_{dfes} to measure the distance from the eigenspace U_{N}, which is a sub-eigenspace of U_{All}.



(a) case 1 (b) case 2

Fig. 3.13. Possible distributions of native and non-native samples in the eigenspace U_{All}

The syllable-level mispronunciation detection is an extension of the N/NN classification applied to syllables. Each "test syllable" is treated like a "test word" in step 2, and mispronunciation can be detected in the syllables using the same N/NN classification approach.

3.4.2 System Design and Implementation

Having reviewed the application of PCA in face recognition and how it can be applied to mispronunciation detection, the following section will discuss the implementation of PCA-based mispronunciation detection in detail, including database construction, data pre-processing, feature selection, eigenspace training, and detection threshold optimization.

Database Construction

Although the algorithm is not language specific, Spanish was chosen for this work. The database used in this work is relatively small and contains only 10 Spanish words listed in Table 3.2. These words were selected by language experts to cover a variety of common mispronunciations exhibited by American speakers learning Spanish. There were 13 male speakers, seven of them native speakers and six non-native. Each speaker repeated each of the 10 words five times.

Because of the limited size of the database, the Leave-One-Out method was used for training and testing and is further discussed in Section 3.4.3.

Table 3.2 List of Spanish words and syllables

Words	**Syllables**	**Words**	**Syllables**
`jamaica`	/ja/, /mai/, /ca/	`tordurados`	/tor/, /tu/, /ra/, /do/, /s/
`tres`	/tr/, /e/, /s/	`accidente`	/ac/, /ci/, /den/, /te/
`gemelas`	/ge/, /me/, /la/, /s/	`construccion`	/cons/, /truc/, /cion/
`hierro`	/hie/, /rro/	`puertorriquena`	/puer/, /tor/, /ri/, /que/, /na/
`pala`	/pa/, /la/	`aire`	/ai/, /re/

Data Pre-processing and Feature Extraction

The PCA method requires centralized and uniform-size input features for training and testing. For centralization, periods of silence before and after the actual speech segment were removed using voiced/unvoiced detection. To make all samples uniform in size, the samples were time-scale modified to match the average duration of the training data. To detect mispronunciation in syllable level, words are divided into syllable using Dynamic Time Warping (DTW). Furthermore, to improve the detection performance, the background noise was suppressed and the amplitude of each sample was normalized to unity.

Spectrograms and MFCCs were chosen as input features to the detection system. These features were computed for frames with window size 25 ms (using a Hamming window) and 15 ms overlap between frames. The spectrogram feature space is 50-dimensional with each dimension spanning 320 Hz to 16 KHz. The MFCC feature space was 13-dimensional representing the first 13 Mel-scale cepstral coefficients.

Eigenspace Training and Detection Threshold Optimization

As discussed in Section3.4.1, these three steps are: (a) word verification; (b) N/NN classification; and (c) syllable-level mispronunciation detection. The detection system runs on a word-by-word basis. For each word, W_{i^*}, $i^* = 1, 2, ..., 10$, each step (a), (b), and (c) undergoes the following 3 phases of training.

- Phase 1: Train to determine eigenspace U.
- Phase 2: Compute two sets of distances e^1_{dfes} and e^2_{dfes}, where e^1_{dfes} corresponds to the distances from "class 1" samples to the eigenspace U and e^2_{dfes} corresponds to the distance from "class 2" samples to U.
- Phase 3: Find an optimal detection threshold T that separates these two sets of distances.

Even though the three training phases are the same for the three steps (a), (b), and (c), each step has a different trained eigenspace, uses different class 1 and class 2 data, and generates different thresholds. Table 3.3 summarizes the differences in the three training phases of the three steps. In Table 3.3, the data used in training to obtain the eigenspace (second row) and to compute e^1_{dfes} (third row) are slightly different. Even though they share the same notation in the table, they represent two disjoint subsets within the class 1 data. Furthermore, different eigenspaces are trained using the Leave-One-Out approach. In this approach, samples corresponding to one speaker are left out. The samples from remaining speakers are used to train the eigenspace, and the samples from the left out speaker are used to find e^1_{dfes}. This is repeated for all speakers belonging to class 1. As mentioned, this approach leads

to several trained eigenspaces. The e^2_{dies} distances are then computed for class 2 data to those different trained eigenspaces and averaged for each speaker in class 2. The distributions formed by e^1_{dfes} and e^2_{dfes}, an example shown in Figure 3.14, are then used to find the optimal threshold. By convention [32], the dimension of the eigenspace in each step is determined by selecting eigenvectors that represent 80% variance of the total principle components. For example, the dimension of the eigenspace used to determine e_{dfes} in Figure 3.14 is 18.

Table 3.3 Data and eigenspace comparison of 3 mispronunciation detection steps

	Step 1	**Step 2**	**Step 3**
Eigenspace	$U_{\text{All}}(W_{i^*})$	$U_{\text{N}}(W_{i^*})$	$U_{\text{N}}(W_{i^*}, k)$
Data used to train eigenspace	$\mathcal{D}_{\text{All}}(W_{i^*})$	$\mathcal{D}_{\text{N}}(W_{i^*})$	$\mathcal{D}_{\text{N}}(W_{i^*}, k)$
Class 1 data used to find e^1_{dfes}	$\mathcal{D}_{\text{All}}(W_{i^*})$	$\mathcal{D}_{\text{N}}(W_{i^*})$	$\mathcal{D}_{\text{N}}(W_{i^*}, k)$
Class 2 data used to find e^2_{dfes}	$\mathcal{D}_{\text{All}}(W_i), i = 1, 2, ..., L,$ but $i \neq i^*$	$\mathcal{D}_{\text{NN}}(W_{i^*})$	$\mathcal{D}_{\text{NN}}(W_{i^*}, k)$

Note: U denotes eigenspace, $\mathcal{D}$ denotes data, All denotes both native and non-native included, N denotes native, NN denotes non-native, L denotes the number of different words, (W_{i^*}, k) denotes k^{th} syllable in word i^*, and $k = 1, 2, ..., K(i^*)$ where $K(i^*)$ is the number of syllables in word $W(i^*)$

The main goal of the training process is to find the "optimal threshold" to detect mispronunciations in the test samples. In each step, since the test samples come from two classes of data which are supposed to be separated, the optimal threshold T should be the one that separates these two classes best. This is done using Bayes rule by finding the optimal threshold to separate two sets of distances e^1_{dfes} for class 1 and e^2_{dfes} for class 2 data. Let random variables X_1 and X_2 represent e^1_{dfes} and e^2_{dfes} and assume they are both Gaussian distributed with prior probabilities $P(\omega_1)$ and $P(\omega_2)$,

where ω_1 and $\omega_2$denote the classes of these two groups. The classification error can be computed by Equation (3.9) using Bayes rule

$$P(\text{error}) = \int_{x^*}^{\infty} p(x \mid \omega_1)P(\omega_1)\mathrm{d}x + \int_{-\infty}^{x^*} p(x \mid \omega_2)P(\omega_2)\mathrm{d}x, \tag{3.9}$$

where the optimal threshold x^* or T can be found by computing the discriminant function $g(x)$ at $g(x) = 0$ where

$$g(x) = p(x \mid \omega_1)P(\omega_1) - p(x \mid \omega_2)P(\omega_2). \tag{3.10}$$

Figure 3.14 illustrates the numerical distribution of e^1_{dfes} (top line) and e^2_{dfes} (bottom line) as an example of the word verification step for the word `tres`. The distances are averaged at five repetitions for each speaker. Figure 3.15 plots the corresponding theoretical Gaussian distribution in order to obtain a more reliable optimal threshold. MFCCs were used as the feature set here. Note that since there are 9 words from class 2 and only one word from class 1, distance e^1_{dfes} has been repeated nine times to make them comparable to the e^2_{dfes}. The optimal threshold T_d can be found by computing the minimum error rate of the theoretical distribution (Gaussian distribution assumed) shown in Figure 3.15. By Bayes rule, the threshold is optimal when it provides the theoretical minimum classification error $P(\text{error})$ (0.030% in this case). The $P(\text{error})$ computed using spectrograms is similar but a little higher (0.447%).

Fig. 3.16 illustrates the native/non-native distribution of data in N/NN classification step for the word `pala`. The $P(\text{error})$ obtained using MFCCs (Figure 3.17) and spectrograms are 0.001% and 11.65% respectively. Experimental results shows that MFCCs are much better than spectrograms in separating data.

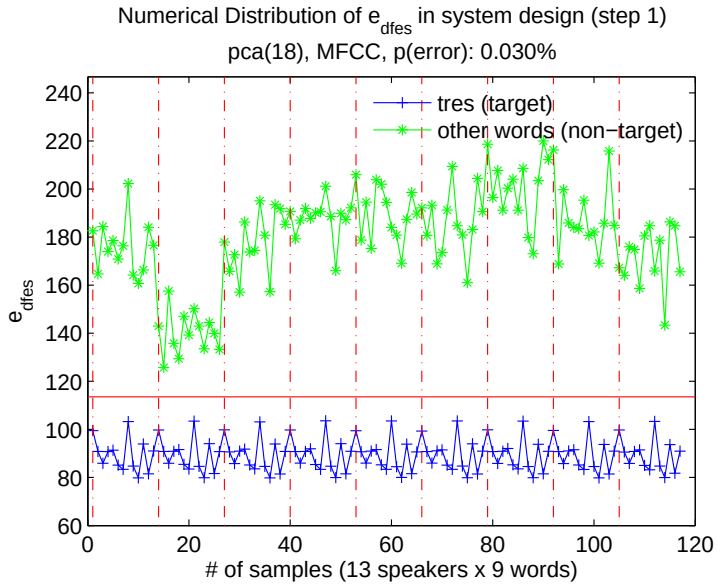


Fig. 3.14. Numerical distribution of class 1 and class 2 data in word verification (target word: `tres`, feature: MFCCs)

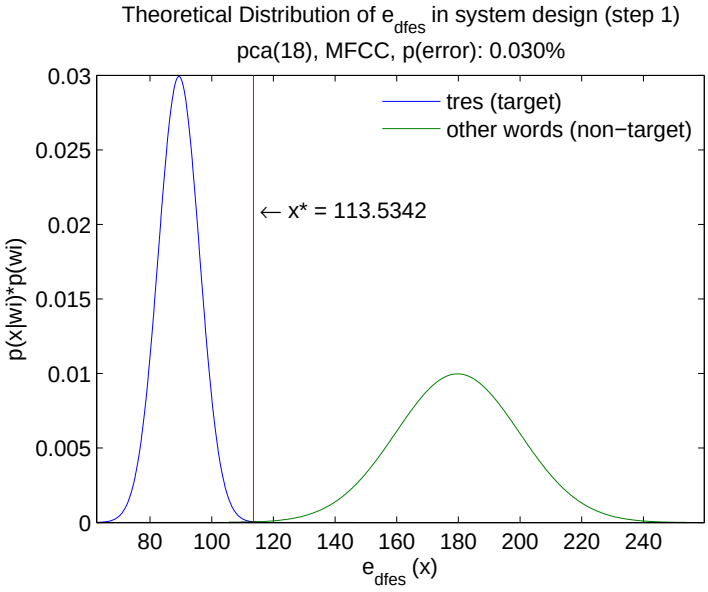


Fig. 3.15. Theoretical distribution of class 1 and class 2 data in word verification (target word: `tres`, feature: MFCCs)

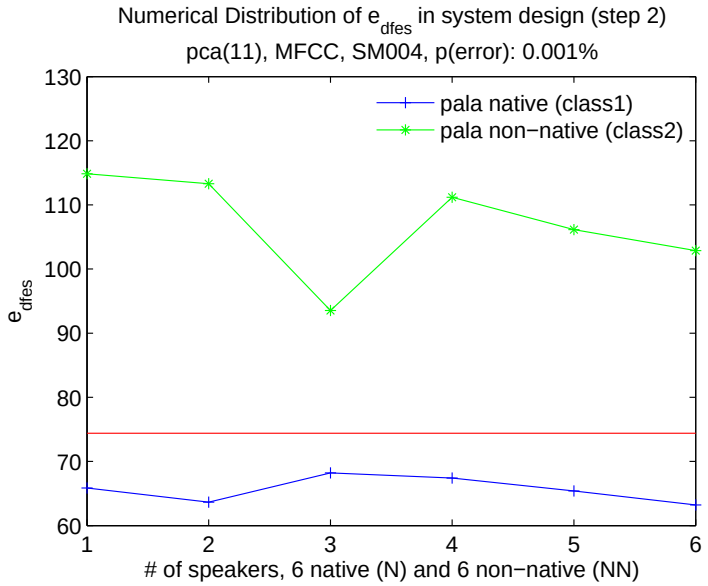


Fig. 3.16. Numerical distribution of e_{dfes} of the class 1 and class 2 data in N/NN classification (target word: `pala`, feature: MFCCs)

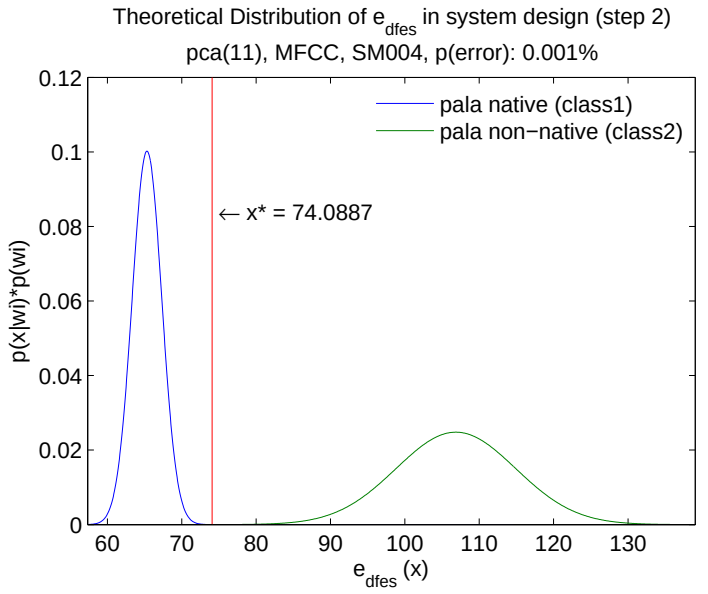


Fig. 3.17. Theoretical distribution of e_{dfes} of the class 1 and class 2 data in N/NN classification (target word: `pala`, feature: MFCCs)

3.4.3 System Testing and Results

After eigenspace training and detection threshold optimization, the mispronunciation detection system is built up. The following section presents the system testing results in each step.

Leave-One-Out Training and Testing

Because of the relatively small size of the database, the Leave-One-Out (LOO) method is used for training and testing. Traditionally in LOO, all but one sample is used in training and the left out sample is used for testing. In our case, samples belonging to one speaker (i.e. all five repetitions) are left out for testing and all samples belonging to all other speakers are used in system training, which includes the three phases discussed in the Section 3.4.2: eigenspace construction (U), 2-class distance measurement (e^1_{dfes},e^2_{dfes}), and detection threshold optimization (T_d, T_c and T_k).

For example, in the word verification step, the goal is to verify whether or not the "test word" is the target word W_{i^*}. The number of samples available for training and testing in this step is 650 (10 different words $\times$ 13 speakers $\times$ 5 repetitions). Using the LOO method, five repetitions of word W_{i^*} from one speaker are left out for testing and the remaining 645 samples are used to train the system and obtain the optimal threshold T_d. This is repeated for each speaker. At the end, there are 130 trained system/threshold combinations and five test samples for each system to be validated.

In the native/non-native classification, the number of samples available for training and testing is 65 (13 speakers $\times$ 5 repetitions of the target word). Five repetitions from one speaker are left out for testing and the remaining 60 samples are used to train the system and obtain the optimal threshold T_c. At the end, there are 13 system/threshold combinations and five test samples for each system to be validated.

Results of Word Verification and N/NN Classification

Compared with the theoretical error rate in Equation (3.9), the performance of the mispronunciation detection system is measured by the numerical error rate P_e

$$P_e = \frac{N_{e1} + N_{e2}}{N_1 + N_2} \ , \tag{3.11}$$

where N_1, N_2 are the number of test samples from class 1 and 2, and N_{e1}, N_{e2} are the number of misclassified samples from each class.

In word verification, the error rate P_e is always below 3% and 7% for MFCCs and spectrograms respectively. In N/NN classification, the performance based on HMMs using MFCCs is also compared along with the PCA method. Figure 3.18 and 3.19 show the results of the Leave-One-Out method applied to the word `aire` using PCA and HMMs. The 13 columns in the figure represent the 13 speakers of the word `aire`. For each column (speaker), there are five samples, which are compared against the threshold. The bottom line corresponds to the seven native speakers and the top line corresponds to the six non-native speakers. The thresholds during each LOO iteration vary slightly because of small differences in the training database during each trial. These variations diminish as the database size increases. For the PCA method using MFCCs, there is one sample from both native speaker 4 and 6 in Figure 3.18 that are slightly above the threshold and misclassified as non-native and all the rest from both classes are correctly classified. A similar situation is illustrated in Figure 3.19 with slightly higher P_e. Complete word verification and N/NN classification results of all 10 words are provided in Table 3.4.

In Table 3.4, MFCCs are shown to perform much better than spectrograms using PCA, especially in step 2, N/NN classification. The PCA method performs slightly better on average than HMMs in this data-limited case. However, general parameters of HMMs are used and they are not optimized for this specific system. With respect to the computational cost, training an eigenspace is almost 10^3 times faster than training HMMs (5 ms vs. 5 s), while in sample testing, the PCA method is also 60-80 times faster (about 4 ms vs. 250 ms) than HMMs depending on the length of the word.

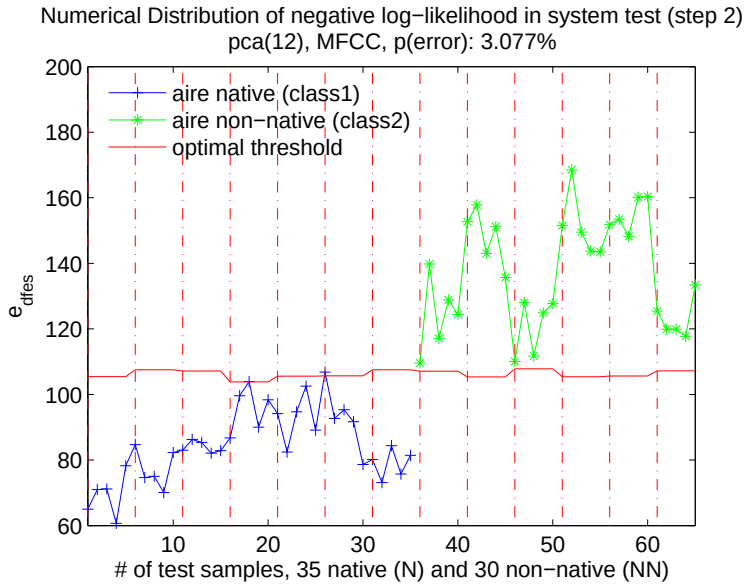


Fig. 3.18. Numerical distribution of e_{dfes} of the test samples in N/NN classification using PCA (target word: `aire`, feature: MFCCs)

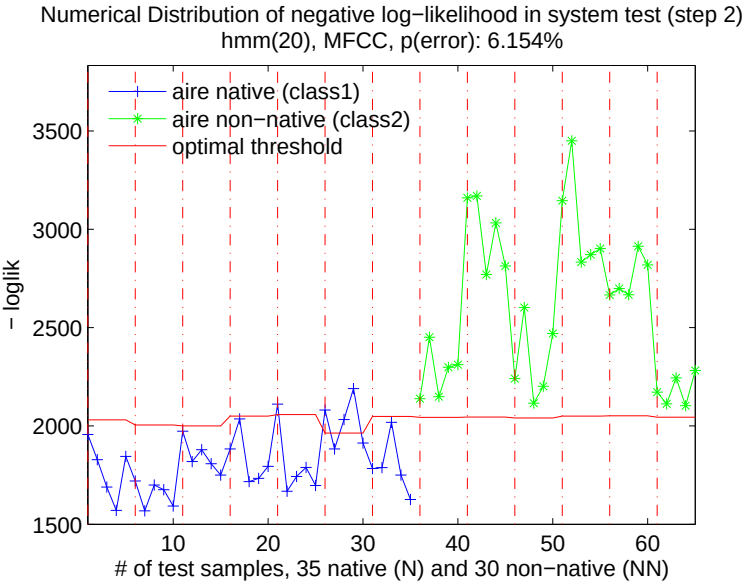


Fig. 3.19. Numerical distribution of negative log-likelihood of the test samples in N/NN classification using HMM (target word: `aire`, feature: MFCCs)

Results of Syllable-Level Mispronunciation Detection

For syllable-level mispronunciation detection, an approach similar to the N/NN classification is followed, except that it is done at a syllable level. Though in real

Table 3.4 Error rate P_e in word verification and N/NN classification

Steps	Methods -Features	Words (%) W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	Max	Min	Avg.
1	PCA-MFCCs	2.04	0.03	0.63	0.07	0.19	0.18	0.37	1.04	0.08	1.43	2.04	0.03	**0.61**
	PCA-Spec.	6.70	0.45	1.56	2.67	5.40	3.47	3.29	5.24	2.22	6.67	6.70	0.45	**3.77**
2	PCA-MFCCs	12.31	10.77	1.54	12.31	7.69	7.69	6.15	9.23	4.62	3.08	12.31	1.54	**7.54**
	PCA-Spec.	24.62	16.92	27.69	21.54	18.46	18.46	10.77	18.85	13.85	15.38	27.69	10.77	**18.15**
	HMM-MFCCs	3.08	10.77	6.51	9.23	10.77	12.31	9.23	7.69	4.62	6.154	12.31	3.08	**8.04**

applications, only samples that are classified as non-native in the N/NN classification would be processed by the 3rd step, to make testing results comparable and unbiased, all test samples including native samples are also used here for evaluation.

There is a major difference between this step and the previous two steps. This is because the assumption that two classes of data in eigenspace training and e_{dfes} computing are separated is no longer valid, namely, some syllables may be pronounced well enough that they cannot be used to distinguish native and non-native. If this is the case, the threshold obtained using Bayes rules is biased and moves towards the native class, which dramatically increases the classification error rate P_e. Thus, P_e cannot be used to measure the performance of mispronunciation detection. Instead, it serves as the similarity measurement of the syllables pronounced by both native and non-native classes. By dividing the total error rate P_e into False Negative Rate (FNR) and False Positive Rate (FPR) in Equation (3.12),

$$\text{FNR} = \frac{N_{e1}}{N_1} \quad \text{and} \quad \text{FPR} = \frac{N_{e2}}{N_2} \tag{3.12}$$

it is easy to find that with a relatively low FNR, higher FPR indicates better pronunciation of the syllable, while lower FPR shows the problematic syllable that one should pay attention to.

Fig. 3.20 and 3.21 illustrate two examples where "good" (FPR = 80%) and "bad" (FPR = 23.3%) mispronunciations are detected. Table 3.5 shows FNR/FPR for each syllable and highlights all FPR $\leq$ 30%. From Table 3.5, common mispronunciation problems [33] like vowel reduction, aspiration, linkage and stress are successfully detected in the test database. Some of them are listed below:

- Vowel Reduction: /pa/ in `pala`; /den/ in `accidente`; /ie/ in `arie`
- Aspiration: /pa/ in `pala`; /puer/ in `puertorriquena`
- Linkage: /tr/ in `tres`; /cons/ in `construccion`; /na/ in `puertorriquena`
- Stress: /ci/ in `accidente`; /cons/ and /cion/ in `construccion`

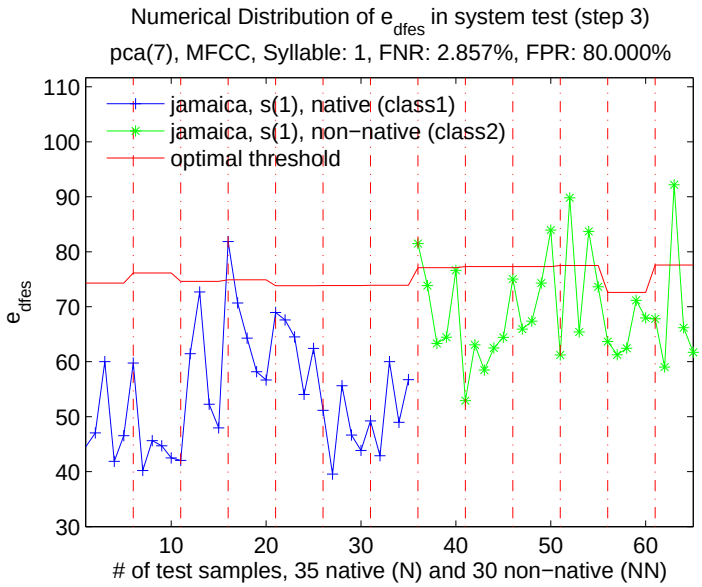


Fig. 3.20. Numerical distribution of e_{dfes} of the test samples in mispronunciation classification using PCA (target syllable: `jamaica`, /ja/, feature: MFCCs)

Table 3.5 FNR and FPR in mispronunciation detection

Words	Syllables	FNR*	FPR**	Words	Syllables	FNR	FPR
jamaica	/ja/	2.86%	80.00%	pala	/pa/	2.86%	**23.33%**
	/mai/	0.00%	80.00%		/la/	0.00%	80.00%
	/ca/	11.43%	53.33%	accidente	/ac/	5.71%	53.33%
tres	/tr/	5.71%	**16.67%**		/ci/	2.86%	**26.67%**
	/e/	0.00%	73.33%		/den/	5.71%	**10.00%**
	/s/	11.43%	36.67%		/te/	8.57%	66.67%
gemelas	/ge/	5.71%	40.00%	construccion	/cons/	0.00%	**30.00%**
	/me/	2.86%	43.33%		/truc/	2.86%	40.00%
	/la/	2.86%	80.00%		/cion/	8.57%	**30.00%**
	/s/	5.71%	60.00%	puertorriquena	/puer/	5.71%	**30.00%**
hierro	/hie/	2.86%	56.67%		/tor/	2.86%	80.00%
	/rro/	8.57%	40.00%		/ri/	0.00%	70.00%
torturados	/tor/	8.57%	**30.00%**		/que/	2.86%	70.00%
	/tu/	2.86%	43.33%		/na/	2.86%	**26.67%**
	/ra/	8.57%	70.00%	aire	/ai/	2.86%	33.33%
	/do/	17.14%	73.33%		/re/	0.00%	**30.00%**
	/s/	8.57%	86.67%				

*FNR: False Negative Rate; **FPR: False Positive Rate

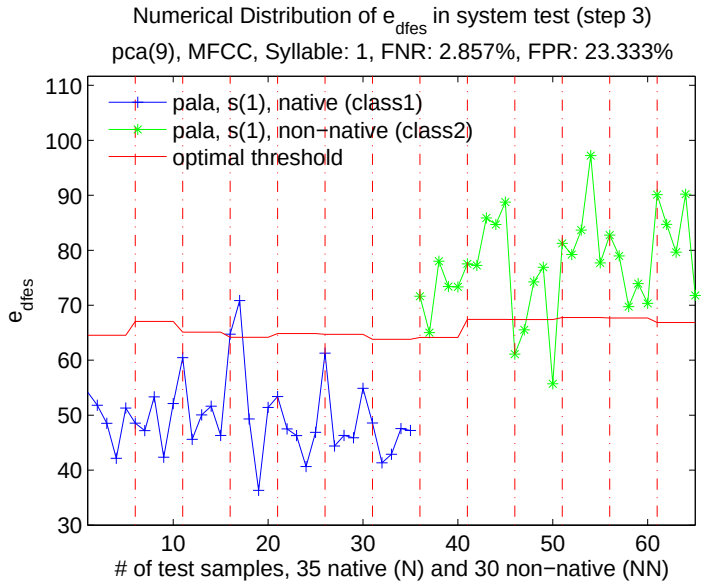


Fig. 3.21. Numerical distribution of e_{dfes} of the test samples in mispronunciation classification using PCA (target syllable: `pala`, /pa/, feature: MFCCs)

4. MISPRONUNCIATION DETECTION FOR SPEECH RECOGNITION ADAPTATION

Mispronunciation detection addressed in this section, is aimed at improving the performance of speech recognition engines, when the users are international and may have a wide variety of accents. Automatic speech recognition is becoming more and more important and challenging as businesses become more global. Two problems of particular interest to us in the mispronunciation work are:

1. Improving name recognition by pronunciation learning, as one of the approaches to adapt speech recognition to the pronunciations of foreign names and accents;
2. Detecting and classifying accents with both acoustic and phonetic information.

Name recognition is quite difficult when the names includes international names and are being pronounced by individuals from all over the world. Thus, we firstly consider the variations in the pronunciations of names and incorporate those variations into the pronunciation dictionary to improve name recognition. The detailed algorithm and procedure is discussed in Section Section 4.1.

Second, we consider the problem of accent detection and classification. The motivation for this study is that a good solution to the detection/classification problem could dramatically improve the performance of name recognition algorithms. In this study, we start with a baseline accent classification performance using Gaussian Mixture Model (GMM) classifier with discriminative Mel-Frequency Cepstral Coefficients (MFCCs) speech feature with Heteroscedastic Linear Discriminant Analysis (HLDA). Then, we use phonetic information, such as the five fundamental vowel shifts in each type of accented speech for better classification. The specifics are presented in 4.2.

4.1 Name Recognition Improvement by Pronunciation Learning

Automatic name recognition is a challenging problem in Automatic Speech Recognition (ASR), since the spelling and pronunciation of names are normally significantly different compared with other normal speech. In addition, the speaker pronouncing the name with an accent. For example, an English speaker with a Chinese accents pronouncing an uncommon English name, or a native American English speaker pronouncing a Chinese name written in English, will often cause pronunciation discrepancies and decrease the name recognition accuracy.

To improve the name recognition performance, there are three important issues that addressed, the recognition algorithm, the acoustic modeling and the lexical modeling. Regarding the recognition algorithm, the HMM-based speech recognition algorithm is very commonly used for recognizing names, keywords and special terms [34], [35]. Other methods such as Deep Neural Networks (DNN) [36], [37] and Support Vector Machine (SVM) [38] [39], and hybrid approaches are also popular in the area of speech recognition. In addition, some clustering and adaptation techniques, such as speaker clustering and massive adaptation to matching testing data with training data, has been shown to improve name recognition accuracy [40].

To improve the acoustic modeling, various statics modeling methods have been explored such as Maximum Likelihood Linear Regression (MLLR) [41] and Maximum a Posterior (MAP) algorithms [42]. Bouselmi et al., [43] used both MLLR and MAP to adapt English acoustic models with Japanese accents and claimed to achieve more than 50% improvement in name recognition. However, most of the acoustic modeling required obtaining either the language origin of the name to be pronounced or the nationality of the speaker, in order to adapt the name recognition.

To improve the lexical modeling, one can combine the Spelling-to-Pronunciation rules of different languages, in order to recognize a foreign name pronounced by a foreigner, such as a French name pronounced by a French speaker [44], [45]. Or one can learn the variation in pronunciation that is not covered in the original pronunciation

dictionary, through training data and update the pronunciation dictionary and lexicon model accordingly [46].

There are at least three reasons why pronunciations of certain names are not covered in the original pronunciation dictionary:

1. Foreign names may have different letter-to-phoneme interpolation rules which are uncommon in English-based dictionaries;
2. People of various nationalities may tend to pronounce names differently;
3. Some of the names may have uncommon or multiple pronunciations and may not be fully covered in the main dictionary.

This work mainly focuses on the 3rd aspect to improve the lexical modeling [1]. The proposed pronunciation learning algorithm is similar to [46] , but with more flexibility in pronunciation searching (over 10,000 candidate pronunciations per word on average vs. 150). In the post-learning pruning phase, the algorithm address the potential overlap in pronunciation space of similar names and words, in addition to avoiding pronunciation overfitting. From a practical perspective, this algorithm is not designed to find phone-level scores and indicate changes to phonemes in pronunciation. Rather, it efficiently tests the performance of candidate pronunciations and selects the best of them. Once the pronunciations are learned and pruned, the pronunciation dictionary (which originally contained phoneme attributes of American English and pronunciations of common American English names), is updated. Significant improvement is achieved with the new lexicon model.

4.1.1 Overview of Name Recognition

The grammar-based name recognizer used in this project is developed by the speech team in ININ. Besides the audio data containing pronunciations of names, it

[1]This project is sponsored by Interactive Intelligence (ININ) Incorporation to improve the name recognition in the Interactive Voice Response (IVR) system of customer support.

requires three other input components: the lexicon model, the acoustic model and the name grammar specification.

The acoustic model models each phoneme in the whole phoneme space and make them distinguishable from each other through Maximum Likelihood Estimation (MLE) training. The ININ name recognizer uses even more advanced discriminative training based on Linear Discriminative Analysis (LDA). Mel-Frequency Cepstral Coefficients (MFCCs) are used as acoustic feature for training with Cepstral Mean and Variance Normalization (CMVN).

The lexicon model provides the reference pronunciation in the form of phoneme sequences for names to be recognized. It includes the prototype phoneme dictionary, which specifies the attributes of 39 Arpabet phonemes [47], one or more pronunciation dictionaries, Spelling-To-Pronunciation (STP) interpreter, and text normalizer. The pronunciation dictionaries cover most common words with their associated pronunciations. For names that are uncommon, the STP interpreter will interpret the pronunciations from the word spelling.

Table 4.1 shows the beginning part of the prototype pronunciation dictionary, where each entry is a line of text with one word followed by its corresponding phoneme sequence illustrating its pronunciation. Some of the words contains multiple pronunciations such as `a` and `abbe`. There are 31306 entries (pairs of word and its pronunciation) in the prototype pronunciation dictionary used in this project.

Grammar Specification specifies all names that can be recognized using the name recognizer. Names outside the grammar specification can only be recognized as one of the names included in the grammar. Generally, the more names included, the more challenging it is to correctly recognize each name, since the confusion between names will rise.

Figure 4.1 demonstrates the relationship among various components of the name recognition system. As mentioned above, acoustic model, lexicon model and name grammar specification are the 3 major input components. With input audio utterances, the grammar-based recognizer outputs recognition results, which are evaluated

Table 4.1 An example of pronunciation dictionary

a ah
a ey
aardvark aa r d v aa r k
abacus ae b ah k ah s
abalone æb ah l ow n iy
abandoned ah b ae n d ah n d
abandoning ah b ae n d ah n ih ng
abashed ah b ae sh t
abates ah b ey t s
abbe ae b iy
abbe ae b ey
abbott ae b ah t
...

along with the transcription using scoring tools, such as "sclite" from National Institute of Standards and Technology (NIST). The name of input utterance to the name

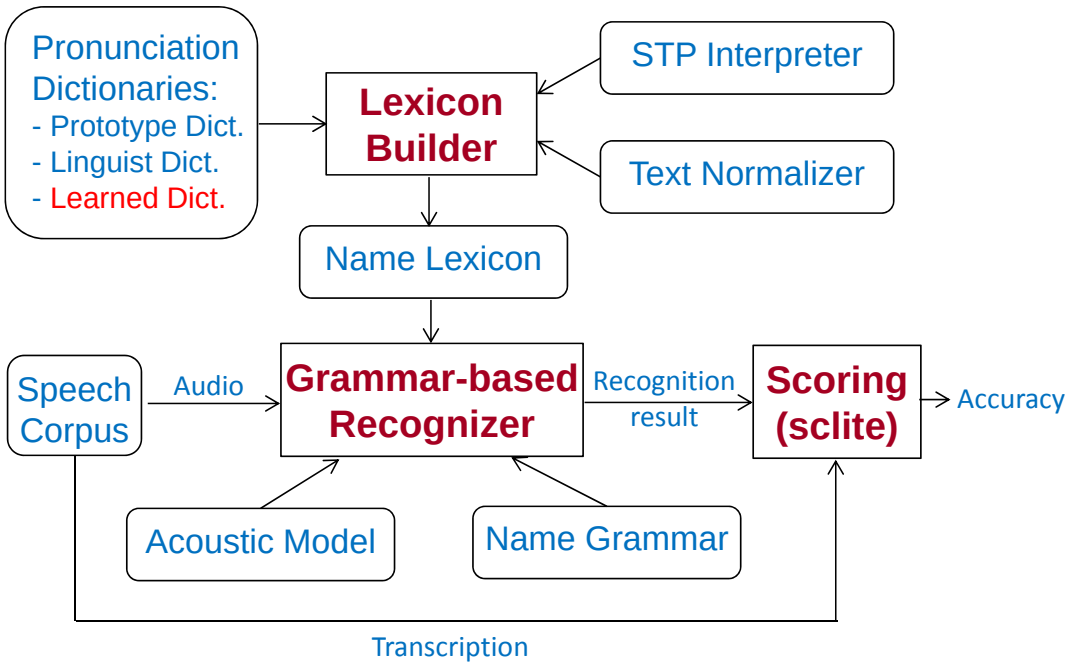


Fig. 4.1. Structure of name recognition

recognizer is called the reference name, $\mathcal{N}_{\text{REF}}$, and the name that the name recog-

nizer picked as output is called the hypothesis name, $\mathcal{N}_{\text{HYP}}$. $\mathcal{N}_{\text{HYP}}$ comes with a Total Confidence Score (TCS), which shows how confident the recognizer is that $\mathcal{N}_{\text{HYP}}$ is correct. This score can be translate into Word Confidence Score (WCS), which indicates the word-level confidence. TCS and WCS can be formulated as shown in Equation (4.1),

$$TCS = \frac{P(O_{\mathcal{N}}|\mathcal{N})}{P(O_{\mathcal{N}})},\ WCS = \frac{P(O_{\mathcal{W}}|\mathcal{W})}{P(O_{\mathcal{W}})},\ TCS, WCS \in [0,1] \tag{4.1}$$

where $P(O_{\mathcal{N}})$ and $P(O_{\mathcal{W}})$ are the probabilities of name-level and word-level observation, and $P(O_{\mathcal{N}}|\mathcal{N})$ and $P(O_{\mathcal{W}}|\mathcal{W})$ are the probabilities of their observations given the name or word respectively.

4.1.2 Name Database and Baseline Performance

The baseline performance of the name recognizer has been tested using two different ININ name databases: pilot and phase-1. The pilot database is a small name database for trial experiments, and the phase-1 name data is one of the two subsets of the most up-to-date ININ name database. Below is the baseline performance with these two databases. The Sentence Error Rate (SER), i.e. Name Error Rate (NER) and Word Error Rate (WER) are computed using the NIST scoring tool sclite.exe.

Table 4.2 Baseline performance in database with different vocabulary size

Database	Grammar Size	No. of Unique Names (Incorrect)	No. of Name Instances (Incorrect)	SER (%)	WER (%)
pilot	610	586 (101)	1940 (167)	7.01	7.02
phase-1	13875	12419 (5307)	38806 (8083)	20.83	17.96

The performance of name recognition is inversely proportional to the grammar size. To visualize this relationship, the whole phase-1 data is divided into a series of subsets $S_{1000}, S_{2000}, \cdots, S_{13000}, S_{13875}$, with different grammar sizes, $1,000, 2,000, \cdots$, up to $13,000$ and $13,875$ respectively. The names in each subset are randomly chosen

from the phase-1 name database, and the larger subset always includes the smaller subset as indicated in Equation (4.2).

$$S_{1000} \subset S_{2000} \subset \cdots \subset S_{13000} \subset S_{13875} \tag{4.2}$$

Then, the name recognition with multi-grammar scales is tested. Figure 4.2 shows how the SER and WER of name recognition smoothly increase when the size of the vocabulary gets larger.

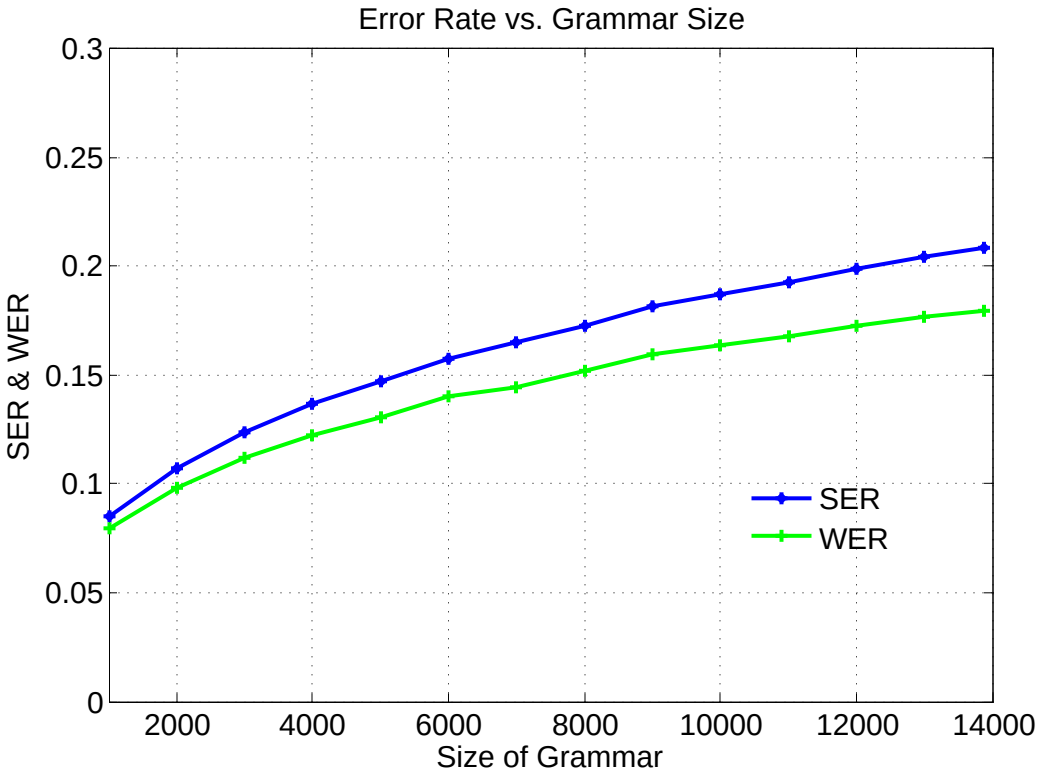


Fig. 4.2. Baseline performance with multiple grammar scales

4.1.3 Pronunciation Learning Algorithm

The fundamental idea of pronunciation learning is straightforward. For each name instance being mis-recognized, search alternative pronunciations that best represent

Table 4.3 Baseline performance with multiple grammar scales

Grammar size	No. of name instances	No. of mis-recognized name instances	SER (%)	WER (%)
1000	2800	239	8.54	7.97
2000	5564	595	10.69	9.81
3000	8329	1032	12.39	11.20
4000	11088	1519	13.70	12.27
5000	13913	2049	14.73	13.07
6000	16659	2629	15.78	14.03
7000	19569	3227	16.49	14.46
8000	22400	3869	17.27	15.24
9000	25116	4568	18.19	15.98
10000	27900	5220	18.71	16.36
11000	30716	5924	19.29	16.78
12000	33472	6661	19.90	17.26
13000	36324	7425	20.44	17.68
13875	38806	8083	20.83	17.96

the realization of that name instance from a list of candidate pronunciations and add it into the learned dictionary if the name instance is successfully recognized with the newly learned pronunciation.

In the real implementation, the pronunciation learning algorithm is applied to each word of the name instance being mis-recognized, rather than the whole name at once. The learned pronunciations will be added to an additional dictionary in the same format as the other existed dictionaries, which is in the format of word-pronunciation pairs. If one pronunciation is learned from one name, it needs to be broken down into words before adding it to the dictionary. More importantly, if the same word appears in different names, it is better to learn the pronunciation of this word without the influence from its adjacent words in different names.

Three issues are important to consider, when designing a pronunciation learning algorithm:

1. How to find similar candidate phonemes that can be used to replace the target phoneme in the reference pronunciation;

2. How to generate the list of candidate pronunciations to be chosen from;
3. How to find the best suitable one from this list for the name instance being worked on.

In the following paragraphs of this section, These issues will be discussed and followed by the discussion of the procedure for pronunciation learning in Section 4.1.4.

Phoneme Similarity Measurement

To measure the similarity between various phonemes, we employ a phoneme alphabet called **Arpabet** and **phoneme confusion matrix**. The phoneme alphabet that serves as the basis for the phoneme confusion matrices and pronunciation dictionary build-up in this project is called Arpabet [47]. It has been used in several speech synthesizers and also used in the Carnegie Mellon University (CMU) Pronouncing Dictionary [48]. The phoneme set contains 39 phonemes designed for speech recognition use.

The acoustic confusion matrix demonstrated in Figure A.1 is a 39×39 symmetric matrix, which shows the acoustic confusion index of each phoneme pair. The linguistic confusion matrix is a binary confusion matrix, shown in Figure A.2. It contains only 0's or 1's to indicate 16 phoneme clusters, as shown in Table 4.4. The phoneme clustering is provided by a group of linguistic experts in ININ based on how phonemes can be confused linguistically when people with different backgrounds speak English. The union of these two confusion matrices, shown in Figure A.3, can be computed by Equation (4.3)

$$d_{\text{union}}(p_i, p_j) = d_{\text{acoustic}}(p_i, p_j) \cdot d_{\text{linguistic}}(p_i, p_j) \tag{4.3}$$

on a phoneme basis, where $d_{\text{acoustic}}(p_i, p_j)$, $d_{\text{linguistic}}(p_i, p_j)$ and $d_{\text{union}}(p_i, p_j)$ are the acoustic, linguistic and unioned confusion indices between phoneme i and j, respectively.

Table 4.4 Linguistic phoneme clustering

Cluster	Phones	Cluster	Phones
1	iy, ih, ay, y	9	k, g
2	ey, eh	10	f, v
3	ae, aa, ao, ah, aw	11	s, z, sh, zh
4	ow, oy	12	ch, jh
5	uw, uh, w	13	m
6	er, r, l	14	n, ng
7	p, b	15	th, dh
8	t, d	16	hh

The reason to use the union of both linguistic and acoustic confusion matrices is that people may alter the phonemes in the reference pronunciation to linguistically or acoustically similar ones. For example, refer to Table A.1, phoneme /ae/ can possibly be pronounced as /eh/ since acoustic similarity, or bas /aa/, /ah/, /ao/ or /aw/, since linguistically they belong to the same phoneme cluster.

After obtaining phoneme confusion indices from the union of both confusion matrices, for each phoneme, we are able to sort the remaining 38 phonemes by their unioned confusion indices with the target phoneme. By dynamically thresholding the sorted the list of the rest 38 phonemes for each target phone, Table A.1 is finally used in this project to generate the candidate phonemes for substitutions in the reference pronunciation. These thresholds are the rounded dividing thresholds (rounded to nearest 0.5) of the nearest cluster, after applying k-means clustering with $k = 4$ to the sorted phoneme list.

Generating Candidate Pronunciations

Given the reference pronunciation of a certain word, $\mathcal{P}_{\text{REF}} = [p_M, p_{M-1}, ..., p_2, p_1]$ in Table 4.5, where M is the length of the pronunciation phoneme sequence, and $N_M, N_{M-1}, ..., N_2, N_1$ are the numbers of phoneme candidates for substitution for

each phoneme in $\mathcal{P}_{REF}$, including the reference phoneme itself provided by Table A.1, the number of candidate pronunciations for this word can be defined as

$$X = \prod_{m=1}^{M} N_m. \tag{4.4}$$

Table 4.5 Definition of $\mathcal{P}_{REF}$ with phoneme substitution candidates

$\mathcal{P}_{\text{REF}}$	p_M	p_{M-1}	...	p_1
Number of Candidates	N_M	N_{M-1}	...	N_1
Index of Candidates	$n_M \in [0, N_M - 1]$	$n_{M-1} \in [0, N_{M-1} - 1]$	...	$n_1 \in [0, N_1 - 1]$

Define $x, x \in [0, X-1]$ as the index of a candidate phoneme sequence with selection of n_Mth, n_{M-1}th, ..., n_1th phoneme candidates from the list of candidate pronunciations, where $n_M \in [0, N_M - 1], n_{M-1} \in [0, N_{M-1} - 1], ..., n_1 \in [0, N_1 - 1]$ are the indices of selection for each phoneme in $\mathcal{P}_{REF}$. There is a one-to-one correspondence between the index of the candidate phoneme sequence x and its set of indices of selected candidate phonemes $(n_M, n_{M-1}, ..., n_1)$:

$$x \leftrightarrow (n_M, n_{M-1}, ..., n_1). \tag{4.5}$$

Their relationship is formulated in Equation (4.6) and Equation (4.7):

$$n_m = \left[\frac{x - x \bmod (\prod_{i=1}^{m} N_{i-1})}{\prod_{i=1}^{m} N_{i-1}} \right] \bmod N_m, \quad x \in [0, X-1], N_0 = 1 \tag{4.6}$$

$$x = \sum_{m=1}^{M} n_m (\prod_{i=1}^{m} N_{i-1}), \quad n_m \in [0, N_{m-1}], N_0 = 1. \tag{4.7}$$

For example, given the word `paine` with reference pronunciation [p ey n] in Table 4.6, the candidate pronunciations of this word with their indices are listed in Table 4.7.

Table 4.6 $\mathcal{P}_{REF}$ of the word `paine` with its phoneme substitution candidates

$\mathcal{P}_{\text{REF}}(M=3)$	p	ey	n
Number of Candidates (N_m)	2	4	2
Candidate (n_m)	b (0) p (1)	eh (0) ey (1) iy (2) ih (3)	n (0) ng (1)

Table 4.7 Candidate pronunciations of the word `paine` with their indices

x	n_3	n_2	n_1	$\mathcal{P}_{\text{CAN}}$		
0	0	0	0	b	eh	n
1	0	0	1	b	eh	ng
2	0	1	0	b	ey	n
3	0	1	1	b	ey	ng
4	0	2	0	b	iy	n
5	0	2	1	b	iy	ng
6	0	3	0	b	ih	n
7	0	3	1	b	ih	ng
8	1	0	0	p	eh	n
9	1	0	1	p	eh	ng
10	1	1	0	p	ey	n
11	1	1	1	p	ey	ng
12	1	2	0	p	iy	n
13	1	2	1	p	iy	ng
14	1	3	0	p	ih	n
15	1	3	1	p	ih	ng

Single-Grammar Recognition Experiment

Once the candidate pronunciations of the target word are generated based on the list of candidate phonemes shown in A.1, these pronunciations are evaluated through a series of single-grammar name recognition experiments.

In the single-grammar name recognition experiments, there is only one name, i.e. the target name, specified in the grammar. Thus, the output hypothesis name $\mathcal{N}_{\text{HYP}}$ is guaranteed to be the same as the input reference name $\mathcal{N}_{\text{REF}}$, and the single-grammar experiment can be considered as a measurement of how likely the input audio recording (name instance) matches the target name, given the lexicon containing only the pronunciations of that name. With the input audio from only the recordings of the target names, these experiments may be used assess the quality of candidate pronunciations for the target name.

As mentioned in the beginning of Section 4.1.3, the pronunciation learning operates at the word level, namely learning pronunciation of a name word by word. However, the input audio and grammar specifications operate at the full name level. To coordinate this case, when learning the pronunciation of an instance of certain name that includes multiple words, only the pronunciation of the word to be learned will be altered and evaluated with its candidate pronunciations, and the pronunciation of all other words will remain the same in the name-level single-grammar name recognition experiment.

Hierarchical Pronunciations Learning

For optimizing the recognition efficiency, when determining the hypothesis result, the ININ name recognizer selects the name whose pronunciation (phoneme sequence) leads to the highest Total Confidence Score (TCS) from all candidates. However, it is not able to provide pronunciation choice for the output name if that name is associated with multiple pronunciations. Thus, it can only provide us the highest TCS from the "best" pronunciation without explicitly indicating which pronunciation is the best, when it is given a set of candidate pronunciations for testing.

To illustrate this constraint of the ININ name recognizer, we use the following example. Given Y names specified in grammar, each name has X_y reference pro-

nunciations, where $y \in [1, Y]$ is the name index. So the total number of reference pronunciations X is

$$X = \sum_{y=1}^{Y} X_y \geq Y \quad \text{and } X_y \geq 1. \tag{4.8}$$

The recognizer outputs the highest TCS (TCS^*) associated with the "best" pronunciation $\mathcal{P}_{x^*}$ from name $\mathcal{N}_{y^*}$, where

$$x^* = \arg\max_{x}(TCS_x) \tag{4.9}$$

$$y^* = \arg\max_{y}(TCS_x) \tag{4.10}$$

are the indices of the "best" pronunciation and its corresponding name. However, only y^* is provided explicitly as the hypothesis name from the recognizer, while x^* is not. The relationship among name index y, pronunciation index x, and its corresponding TCS (TCS_x) is shown in Table 4.8.

Table 4.8 Correspondence of name index (y), pronunciation index (x) and TCS_x

y	x	TCS_x
1	1	TCS_1
	2	TCS_2
	...	...
	X_1	TCS_{X_1}
2	$X_1 + 1$	TCS_{X_1+1}
	$X_1 + 2$	TCS_{X_1+2}
	...	...
	$X_1 + X_2$	$TCS_{X_1+X_2}$
...	...	...
Y	$\sum_{y=1}^{Y-1} + 1$	$TCS_{\sum_{y=1}^{Y-1}+1}$
	$\sum_{y=1}^{Y-1} + 2$	$TCS_{\sum_{y=1}^{Y-1}+2}$
	...	...
	$\sum_{y=1}^{Y} = X$	TCS_X

In order to find the "best matched" pronunciation x^* associated with the output TCS^* and $\mathcal{N}_{y^*}$, we set up a single-grammar recognition experiment, where only one

name $\mathcal{N}_{y^*}$ is specified in the grammar, and hierarchical pronunciation learning is employed to select phoneme $p(n_m)$ in x^* one by one, where $n_m \in [0, N_m - 1]$ and $m \in [1, M]$ is the index of the phoneme candidate for the mth phoneme unit. Finally, we find the selected pronunciation associated with the output. This process is illustrated in Figure 4.3 by example of learning the pronunciation of the word `paine`.

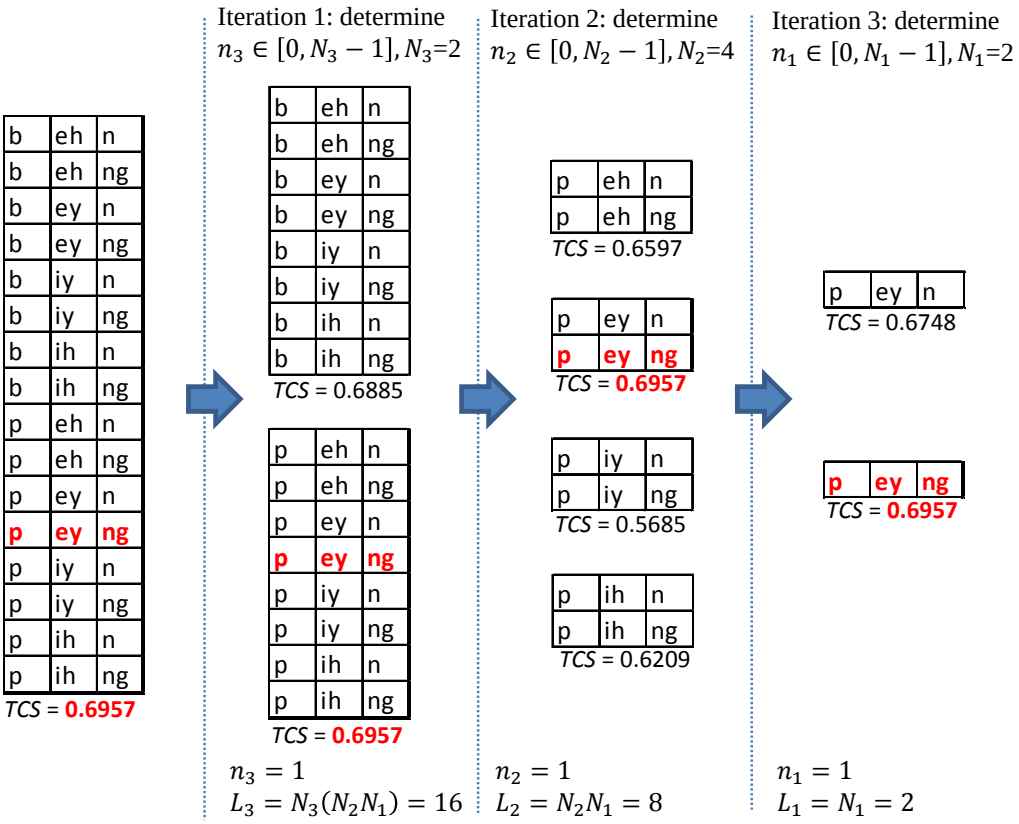


Fig. 4.3. Hierarchical pronunciation learning for word `paine`

As it is indicated in Figure 4.3, the reference and the best matched (learned) pronunciations of word `paine` are $\mathcal{P}_{\text{REF}} = [\text{p ey n}]$ and $\mathcal{P}_{\text{LND}} = [\text{p ey ng}]$ respectively. To track down $\mathcal{P}_{\text{LND}}$, multiple single-grammar experiments with subsets of candidate pronunciations are performed in three iterations, in which the phonemes /p/, /ey/ and /ng/ in $\mathcal{P}_{\text{LND}}$ are determined one by one, by simply tracking the subsets with highest TCS in each iteration. This hierarchical learning approach finds the learned pronunciations with eight runs of the name recognition ($2 + 4 + 2$ runs in three

iterations) and in total process 26 candidate pronunciations of the word `paine` ($16 + 8 + 2$ pronunciations in three iterations), rather than test all 16 pronunciations in 16 runs. Given the computational cost of running the recognizer once is T_1 and processing each candidate pronunciation is T_2, with $T_1 \gg T_2$, this learning algorithm will save more time when the number of candidate pronunciations grows, since it significantly reduces the number of times the recognizer must be run.

In general, based on the definition in Table 4.5, the total number of times one needs to run the name recognizer in hierarchical pronunciation learning can be formulated as $\sum_{m=1}^{M} N_m$. Define

$$L_m = \prod_{i=1}^{m} N_i \tag{4.11}$$

to be the number of pronunciations processed when determining the mth phoneme and

$$L = \sum_{m=1}^{M} L_m \tag{4.12}$$

to be the total number of pronunciations processed while learning the pronunciation for one word. Then total computational cost of the hierarchical pronunciation learning T, which includes the cost of running the recognizer T_{RUN} and the cost of processing the candidate pronunciations T_{PRON}, can be computed as

$$\begin{aligned} T &\approx T_{\text{RUN}} + T_{\text{PRON}} \\ &= \left(\sum_{m=1}^{M} N_m \right) T_1 + \left(\sum_{m=1}^{M} \prod_{i=1}^{m} N_i \right) T_2. \end{aligned} \tag{4.13}$$

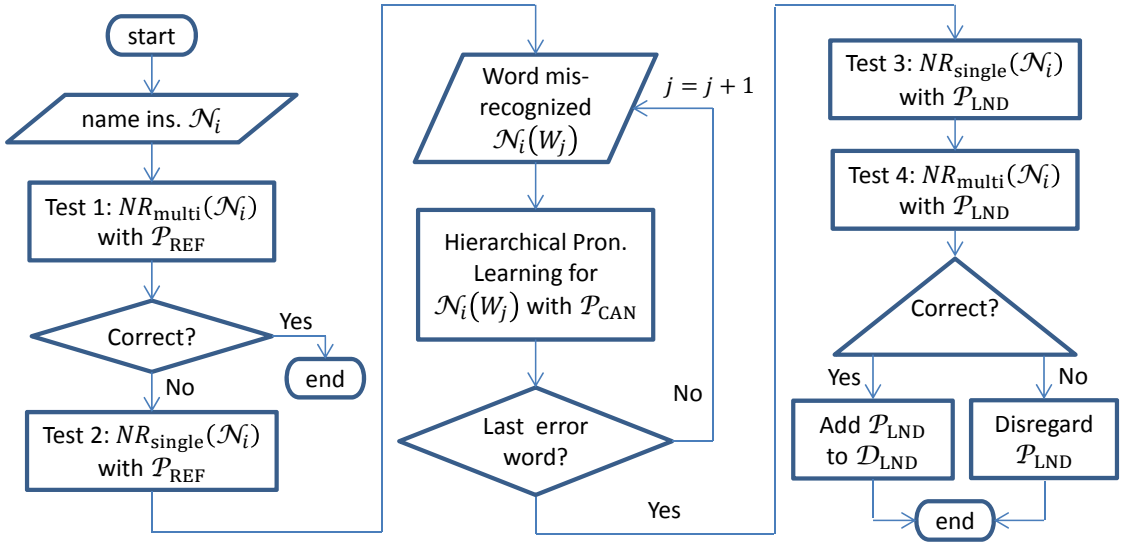
For example, in the pronunciation learning of the word `paine`, the length of the reference pronunciation $\mathcal{P}_{\text{REF}}$ is 3 ($M = 3$) and the numbers of phoneme candidates for each phoneme in $\mathcal{P}_{\text{REF}}$ are $N_3 = 2$, $N_2 = 4$ and $N_1 = 2$. So the total computational cost for learning the word `paine` is

$$\begin{aligned} T_{\texttt{paine}} &= (2 + 4 + 2)T_1 + [(2 \cdot 4 \cdot 2) + (4 \cdot 2) + 2]\, T_2 \\ &= 8T_1 + 26T_2. \end{aligned} \tag{4.14}$$

4.1.4 Pronunciation Learning Procedure

Pronunciations of names are learned instance by instance, rather than name by name, since different instances with recognition errors may be optimized with different candidate pronunciations.

The pronunciation learning procedure is demonstrated in Figure 4.4, which includes four name recognition tests and hierachical pronunciation learning of mis-recognized words. These tests are either single-grammar name recognition (which is only capable of recognizing one specified name) used to measure the accuracy of pronunciations, or multiple-grammar (regular) name recognition. In the second case, all names are used to test if the pronunciation can help to correctly recognize the specified name from a set of all possible names. The results of these tests are listed and compared below in Table 4.9.



$NR_{\text{multi}}(\mathcal{N}_i)/NR_{\text{multi}}(\mathcal{N}_i)$: multiple/single grammar name recognition of the name instance $\mathcal{N}_i$;
$\mathcal{P}_{\text{REF}}$/ $\mathcal{P}_{\text{CAN}}/\mathcal{P}_{\text{LND}}$: reference/candidate/learned pronunciation of the name instance $\mathcal{N}_i$;
$\mathcal{D}_{\text{LND}}$: learned dictionary with all accepted learned pronunciations.

Fig. 4.4. Flowchart of pronunciation learning for mis-recognized name instance

1. **Test 1** performs regular name recognition on current name instance $\mathcal{N}_i$ with all available pronunciations, i.e. reference pronunciations $\mathcal{P}_{\text{REF}}$. It provides a

baseline recognition result, i.e. whether the reference name and hypothsis name are the same ($\mathcal{N}_{\text{REF}} = \mathcal{N}_{\text{HYP}}$), with a total confidence score (TCS) TCS_1 for the name instance $\mathcal{N}_i$. In real practice, this test is not performed for each name instance separately. Instead, all name instances have already gone through the baseline recognition and the name instances with recognition errors have been separated for pronunciation learning.

2. **Test 2** is similar to test 1 and the only difference is that the grammar here contains current name under learning. For single-grammar tests (like tests 2 and 3 here), the reference name $\mathcal{N}_{\text{REF}}$ must be identical to the hypothesis name $\mathcal{N}_{\text{HYP}}$. Test 2 is not required for pronunciation learning and is used to check how challenging it is to recover this name by observing the drop of TCS from TCS_1 to TCS_2.

3. **Test 3** is also optional and is used to check how much the TCS increases from TCS_2 to TCS_3 after learning the most suitable pronunciation for the current name instance $\mathcal{N}_i$.

4. **Test 4** checks if the mis-recognized name instance is recovered by adding the learned pronunciation into the pronunciation dictionary. Usually if $TCS_4 > TCS_1$, the name instance $\mathcal{N}_i$ can be recovered, otherwise, it may not.

Table 4.9 Tests in the process of pronunciation learning for name instance

test	**grammar**	**score**	**recognition result and comparison**
1	multiple	TCS_1	if $\mathcal{N}_{\text{REF}} = \mathcal{N}_{\text{HYP}} \rightarrow$ pass; else $\rightarrow$ continue to test 2
2	single	TCS_2	$\mathcal{N}_{\text{REF}} = \mathcal{N}_{\text{HYP}}$; $TCS_2 < TCS_1$
3	single	TCS_3	$\mathcal{N}_{\text{REF}} = \mathcal{N}_{\text{HYP}}$; $TCS_3 > TCS_2$
4	multiple	TCS_4	if $TCS_4 > TCS_1 \rightarrow \mathcal{N}_{\text{REF}} = \mathcal{N}_{\text{HYP}}$; else $\rightarrow \mathcal{N}_{\text{REF}} \neq \mathcal{N}_{\text{HYP}}$

Here is an example to illustrate the four tests with name `daan greven`, which has five instances misrecognized out of seven in the pilot database. Usually TCS_2 is lower than TCS_1, and that is the reason why a different name other than $\mathcal{N}_{\text{REF}}$ is selected as $\mathcal{N}_{\text{HYP}}$, such as `david record`, `dan joons` and `bill breesmen` here. $TCS_3 - TCS_2$

shows the improvement with the learned pronunciation. If TCS_4 is larger than TCS_1, it is usually equal to TCS_3, indicating the name instance is correctly recognized with the learned pronunciation adding into the pronunciation dictionary.

Table 4.10 An example of pronunciation learning with 4 tests ($\mathcal{N}_{\text{REF}} =$ `daan greven`)

Test	Ins. ID / Speaker ID / Utter. ID	1 100148825 35	2 100148825 17	3 100148825 28	4 100148825 37	5 100148825 8
1	Hyp. name	`david record`	`dan joons`	`dan joons`	`bill breesmen`	`bill breesmen`
	TCS_1	0.610	0.609	0.748	0.656	0.634
	$WCS_1(\mathcal{W}_1)$	0.833	0.910	0.967	0.849	0.845
	$WCS_1(\mathcal{W}_2)$	0.295	0.421	0.399	0.624	0.527
2	Hyp. name	`daan greven`	`daan greven`	`daan greven`	`daan greven`	`daan greven`
	TCS_2	0.567	0.467	0.670	0.614	0.565
	$WCS_2(\mathcal{W}_1)$	0.716	0.773	0.933	0.783	0.801
	$WCS_2(\mathcal{W}_2)$	0.595	0.469	0.586	0.669	0.506
3	Hyp. name	`daan greven`	`daan greven`	`daan greven`	`daan greven`	`daan greven`
	TCS_3	0.814	0.709	0.866	0.834	0.785
	$WCS_3(\mathcal{W}_1)$	0.962	0.937	0.976	0.946	0.937
	$WCS_3(\mathcal{W}_2)$	0.620	0.543	0.703	0.777	0.639
4	Hyp. name	`daan greven`	`daan greven`	`daan greven`	`daan greven`	`daan greven`
	TCS_4	0.814	0.709	0.866	0.834	0.785
	$WCS_4(\mathcal{W}_1)$	0.962	0.937	0.976	0.946	0.937
	$WCS_4(\mathcal{W}_2)$	0.620	0.543	0.703	0.777	0.639
$TCS_1 - TCS_2$		0.044	0.142	0.078	0.042	0.069
$TCS_3 - TCS_2$		0.248	0.242	0.196	0.220	0.220
$TCS_4 - TCS_1$		0.204	0.100	0.118	0.178	0.151

In summary, the flowchart in Figure 4.4 demonstrates a simplified procedure for word-by-word pronunciation learning of names. Details inside this procedure will be discussed later in Section 4.1.5 and Section 4.1.6, such as 1) how to limit the number of learned pronunciations during pronunciation learning; 2) how to deal with names or words with significantly large numbers of candidate pronunciations; 3) how to generate candidate pronunciations for names with multiple reference pronunciations to start with; and 4) how to balance the efficiency and performance by skipping some similar mis-recognized name instance in pronunciation learning.

4.1.5 Pronunciation Pruning

There are two important reasons why the number of learned pronunciations for each name or each word should be limited (or pruned):

1. More pronunciations for a particular name increase the chance of it being correctly recognized. However it may overlap the pronunciations of other names in the name space, hence reduce the opportunity to recognize other names, especially similar names correctly.
2. Learning too many pronunciations for one name using the training data may cause overfitting, which may not generalized well when they are applied to test data.

For the first reason, arbitrarily expanding the dictionary $\mathcal{D}(\mathcal{N}_i)$ of every name from the original reference pronunciation $\mathcal{P}(\mathcal{N}_i)$ without selection, based on the phoneme similarity measurement, has been proved to drop the overall recognition rate significantly, even by a small amount of expansion. The pronunciation of individual name instance is learned independently without being aware of "eating up" the pronunciation space of other similar names in the single-grammar experiments. However, names may become overlapped when adding them all together as an additional dictionary for the recognizer. This issue become more apparent when the grammar size increases.

For the second reason, the issue of overfit become more critical when the training data are too limited, or too many pronunciations are learned from the realizations of the same speakers with the same mis-recognized hypothesis name.

Pruning Approaches

There are three pruning approaches currently implemented in this project:

1. Work from the worst word of the name and add learned pronunciation of words when necessary;

2. Two-metric verification of learned pronunciation, i.e., learning name pronunciation while verifying the effect to similar names;
3. Limit the learned pronunciations for each word in the dictionary.

Approach 1: Skipping Learning Unnecessary Words in Name The name-level learned pronunciation $\mathcal{P}_{\text{LND}}(\mathcal{N}_i)$ operates word by word, and it is the concatenation of word-level learned pronunciations $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$. The first approach sorts $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$ by significance of improving name recognition performance, where $j \in [1, J]$ and J is the number of mis-recognized words in the target name $\mathcal{N}_i$. For each mis-recognized word W_j, the improvement through pronunciation learning is measured by its increase of Word Confidence Score (WCS) from test 2 to test 3, which is formulated in Equation (4.15) as:

$$d_{ij} = WCS_3(\mathcal{N}_i(W_j)) - WCS_2(\mathcal{N}_i(W_j)). \tag{4.15}$$

The increase d_{ij} measures the difference in WCS by replacing $\mathcal{P}_{\text{REF}}(\mathcal{N}_i(W_j))$ with $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$ in $\mathcal{P}_{\text{REF}}(\mathcal{N}_i)$ for name $\mathcal{N}_i$, and hence demonstrates the contribution of $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$. Then, these $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$ are sorted in descending order based on the value of d_j and added into the learned dictionary only when necessary. Taking $\mathcal{P}_{\text{REF}}(\mathcal{N}_i)$ as the initial $\mathcal{P}_{\text{LND}}(\mathcal{N}_i)$, $\mathcal{P}_{\text{REF}}(\mathcal{N}_i(W_j))$ are replaced by $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$ in descending order based on the value of d_j. This process will be terminated once the updated $\mathcal{P}_{\text{LND}}(\mathcal{N}_i)$ helps recognizer correctly recognize $\mathcal{N}_i$ in test 4. The rest of the $\mathcal{P}_{\text{LND}}(\mathcal{N}_i(W_j))$s are disregarded.

For example, the pronunciation learning results of the second name instance in Table 4.10 (with utterance ID 17) are listed in Table 4.11. The reference name $\mathcal{N}_{\text{REF}}$ is `daan greven` and the reference pronunciation $\mathcal{P}_{\text{REF}}$ is {(d aa n), (g r eh v ih n)}.

Since the contribution of the pronunciation learning of the first word (0.164) is larger than the contribution of the second word (0.074), only the learned pronunciation of the first word ($\mathcal{P}_{\text{LND}}(\mathcal{W}_1)$ =(d ae n)) is used to form $\mathcal{P}^*_{\text{LND}}$ in test 3* to test

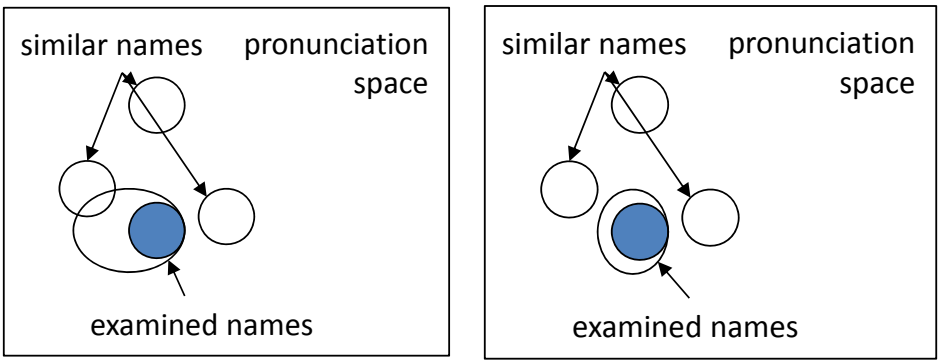
Table 4.11 An example of pronunciation pruning with approach 1

Test	Pronunciation ($\mathcal{P}$)	TCS	$WCS(\mathcal{W}_1)$	$WCS(\mathcal{W}_2)$
2	$\mathcal{P}_{\text{REF}}$ = {(d aa n), (g r eh v ih n)}	0.467	0.773	0.469
3	$\mathcal{P}_{\text{LND}}$ = {(d **ae** n), (g r **ih** v **y ng**)}	0.709	0.937	0.543
3*	$\mathcal{P}^*_{\text{LND}}$ = {(d **ae** n), (g r eh v ih n)}	0.805	0.962	0.595
Improvement with $\mathcal{P}_{\text{LND}}$ ($WCS_3 - WCS_2$)		0.242	0.164	0.074
Improvement with $\mathcal{P}^*_{\text{LND}}$ ($WCS^*_3 - WCS_2$)		0.338	0.189	0.126

if $\mathcal{P}^*_{\text{LND}}$ is good enough to "correct" the misrecognized name. The result shows it is better to keep the pronunciation of the second word original, since the improvement with $\mathcal{P}^*_{\text{LND}}$ is higher than the improvement with $\mathcal{P}_{\text{LND}}$. It is reasonable since the pronunciation of each word is learned individually, and (g r ih v y ng) works best when pronunciation of the first word is (d aa n). When pronunciation of first word is improved to (d ae n), it is better to keep the pronunciation of second word original.

Approach 2: Two-metric Verification of Learned Pronunciation The second approach is a two-metric test. While learning the additional pronunciation of the current target name, which increases the recognition performance on this particular name, it also measures the potential harm to similar names from the learned pronunciation. Figure 4.5 demonstrates the potential risk of pronunciation overlap among different but similar names. The shaded area is the correct region of pronunciation for the examined names. Given S_M is a name set with grammar size M, and all M names are sorted by the similarity to the current target name:

1. Select the most similar m names including the current name itself to compose a name subset S_r, where $r = m/M$ is the portion of names selected. In this project, 10% of the most similar names in the name set are selected ($r = 10\%$);
2. Perform name recognition tests to obtain the name recognition rate NR before (NR_1) and after (NR_2) adding in the learned pronunciation $\mathcal{P}_{\text{LND}}(\mathcal{N}_i)$ of current name instance $\mathcal{N}_i$;



(a) case with overlap (b) case without overlap

Fig. 4.5. Potential pronunciation overlap among different but similar names

3. Add $\mathcal{P}_{\text{LND}}(\mathcal{N}_i)$ into learned dictionary $\mathcal{D}_{\text{LND}}$ only when

$$NR_2(S_r, \cup(\mathcal{D}_{\text{REF}}, \mathcal{P}_{\text{LND}}(\mathcal{N}_i))) > NR_1(S_r, \mathcal{D}_{\text{REF}}), \tag{4.16}$$

where $\mathcal{D}_{\text{REF}}$ is the reference dictionary for current name set S_M.

The similarity between names is measured by the distance in pronunciation space, which will be discussed in the next section.

Approach 3: Limiting the Number of Learned Pronunciations of Words Compared with the second approach, which is in fact an adaptive and intelligent method to calibrate name pronunciations, the third approach defines a firm rule that limits the number of learned pronunciations N for each word in the dictionary. It is because most of the words commonly have very limited variation in pronunciation. In this project, each word allows up to two learned pronunciations ($N = 2$).

The pronunciation of one word may be learned from different name instances that all contain that word. To evaluate the jth learned pronunciation $\mathcal{P}_j(W_i)$ of word W_i, where $j \in [1, J]$ and J is the total number of learned pronunciation for word W_i,

1. first group all name instances $\mathcal{N}_k$, $k \in [1, K]$ that contain the word W_i to form a sub-name set $S(W_i)$;

2. perform name recognition on this subset $S(W_i)$, with word-level learned pronunciation $\mathcal{P}_j(W_i)$, and assign weight $w_k = 1$ to name instance $\mathcal{N}_k$ if it is correctly recognized, or assign $w_k = -1$ to $\mathcal{N}_k$ if it is mis-recognized with $\mathcal{P}_j(W_i)$.

3. take the weighted average of the Total Confidence Score (TCS) over all $\mathcal{N}_k, k \in [1, K]$, as the evaluation score s_{ij} for $\mathcal{P}_j(W_i)$, where

$$s_{ij} = \frac{1}{K} \sum_{k=1}^{K} w_k \, TCS(\mathcal{N}_k(W_i), \mathcal{P}_j(W_i)). \tag{4.17}$$

The evaluation scores s_{ij} for learned pronunciations $\mathcal{P}_j(W_i)$ of word W_i are computed and sorted in descending order. $\mathcal{P}_j(W_i)$ associated with the highest N ($N = 2$ here) weighted average s_{ij} are reserved and the rest are disregarded.

For example, the word `clayton` associates with five name instances of `clayton kie`. So the sub-name set $S(W_i)$ is five name instances pronouncing name `clayton kie`. There are three learned pronunciations: {(k l eh t aw ng), (k l iy k aa n), (k r ih k ae n)} and one of them should be disregarded based on current constraints. To rank these pronunciations, the weighted average of the performance of each pronunciation is calculated using Equation (4.17). The weighted average of the first pronunciation is given by

$$s_1 = (0.725 + 0.674 + 0.788) \times 1 + (0.627 + 0.670) \times (-1) = 0.178 \tag{4.18}$$

and illustrated in Table 4.12.

Table 4.12 An example of pronunciation pruning with approach 3

Ref. Name	Hyp. Name	*TCS*	Weight(w)	Correct?
`clayton kie`	`clayton kie`	0.725	1	Yes
`clayton kie`	`clayton kie`	0.674	1	Yes
`clayton kie`	`wayne loving`	0.627	-1	No
`clayton kie`	`clayton kie`	0.788	1	Yes
`clayton kie`	`craig alvey`	0.670	-1	No

Similarly, the weighted average of the second and third pronunciations are calculated ($s_2 = 0.419$ and $s_3 = 0.141$) and the the third pronunciation (k r ih k ae n) should be removed from the learned dictionary, since it is ranked the third.

In summary, these three pronunciation pruning techniques are described in the same order as they are implemented in the pruning algorithm. The first approach of pruning unnecessary learned pronunciations of words during the pronunciation learning process; the third approach of limiting word-level learned pronunciations is implemented after the learning process, since it requires learned pronunciations of certain word from all name instances containing this word for pruning; the second approach of evaluating learned pronunciations from name instances can be implemented either during or after pronunciation learning from name instances. However, finding similar name and their instances and performing name recognition on these subsets could significantly slow down the learning process, hence this approach is also implemented after pronunciation learning. Table 4.13 summarizes the differences among these methods:

Table 4.13 Comparison of pruning methods with order in the sequence of implementation

	approach	**level**	**timing**	**cost**	**effectiveness**
1	skip unnecessary words in name	word	during learning	low	medium
2	check effect to similar names	name	after learning	high	high
3	limit learned pronunciations of words	word	after learning	low	medium

Distance Measurement in Pronunciation Space

The second pruning approach requires measuring the similarity between different names, in order to select a subset of similar names for pronunciation validation. In this section, the measurement of similarity (distance) between names (their reference pronunciations), based on Dynamic Programming (DP) will be discussed.

Given two names $\mathcal{N}_i$ and $\mathcal{N}_j$ with reference pronunciation $\mathcal{P}_i = [p_1, p_2, \ldots, p_M]$ and $\mathcal{P}_j = [p_1, p_2, \ldots, p_N]$ respectively, the $M \times N$ local distance matrix D stores the confusion values of phoneme pairs (p_m, p_n), where $m \in [1, M]$ and $n \in [1, N]$. To find the distance between these two pronunciations, we first compute the $M \times N$ cumulative distance matrix C from the top-left corner to the bottom-right corner, in which each value $C(m, n)$ shows the shortest cumulative distance from $D(1, 1)$, using a matching template showing where the cumulative distance come from. For example in Figure 4.6, we use the template with location $\{(m-1, n-1), (m, n-1), (m-1, n)\}$ to compute $C(m, n)$, where

$$C(m,n) = \min[\underbrace{C(m-1,n-1)}_{\text{location 1}}, \underbrace{C(m,n-1)}_{\text{location 2}}, \underbrace{C(m-1,n)}_{\text{location 3}}] + D(m,n). \tag{4.19}$$

To compute the boundary values in C, such as $C(m, 1)$ and $C(1, n)$, D should be elongated with the following values:

$$\begin{cases} D(0,0) &= 0 \\ D(m,0) &= \infty \quad m \in [1,M] \\ D(0,n) &= \infty \quad n \in [1,N] \end{cases} \tag{4.20}$$

The trace-back matrix is used to track where the value of $C(m, n)$ comes from, and the right-bottom value in C, i.e. $d_{ij} = C(M, N)$ is the desired distance of these two pronunciations $\mathcal{P}_i$ and $\mathcal{P}_j$. For example in Figure 4.6, $C(M, N) = C(3, 4) = 18$ is d_{ij}, and it can be traced back to $C(1, 1)$ using the trace-back matrix. The path is $C(1, 1) \Rightarrow C(2, 2) \Rightarrow C(2, 3) \Rightarrow C(3, 4)$.

Using this distance measurement, the distance between two different arbitrary names in the name database is computed. In this project, the top 10% of names and of current grammar size are used to compose the subset to validate the learning pronunciation in the second pruning approach.

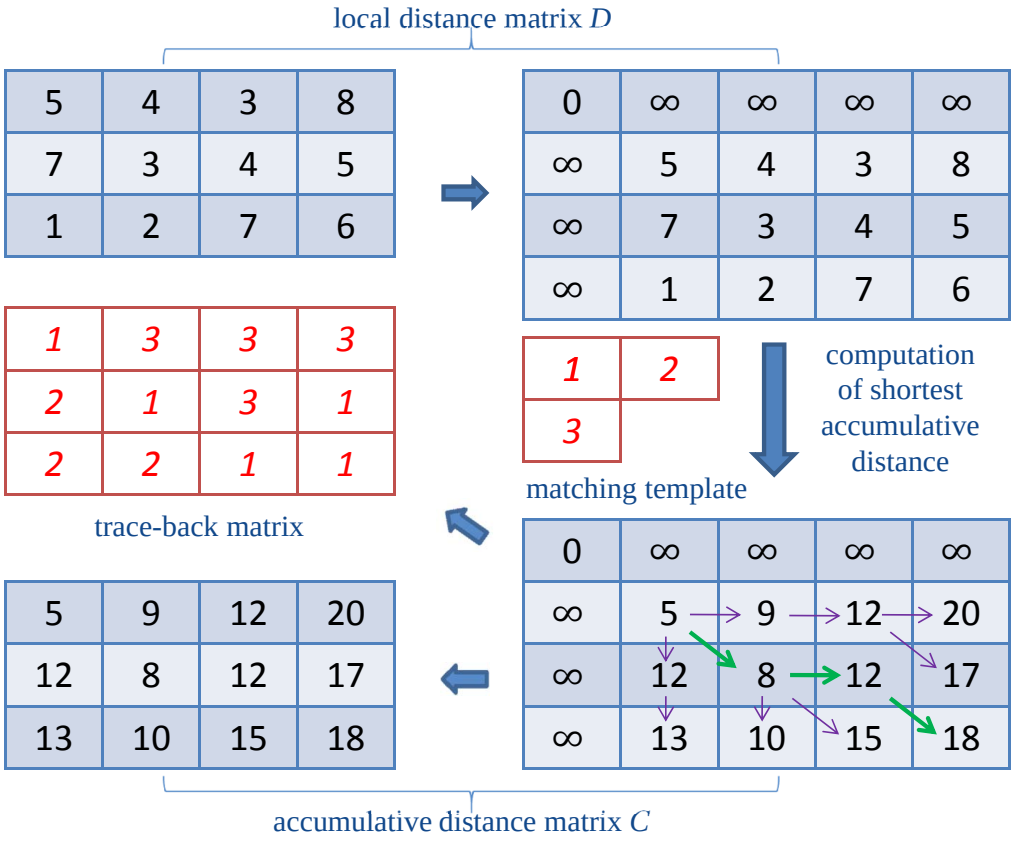


Fig. 4.6. Example of dynamic programming

4.1.6 Implementation and Results

In this section, details during the implementation of pronunciation learning algorithm will be discussed, and then the results through pronunciation learning and performance testing with different sizes of grammar will be provided.

Pronunciation Learning Efficiency Control

The pronunciation learning algorithm can be improved by enhancing the efficiency in searching alternative pronunciations. Here two approaches of efficiency control during pronunciation learning will be discussed:

1. Dynamic threshold on words with large numbers of candidate pronunciations;

2. Optimize the order of phoneme determination by their phoneme candidates.

Based on Table A.1, there is on average four phoneme candidates including the original for each phoneme. Suppose M is the number of phonemes in one reference pronunciation $\mathcal{P}_{\text{REF}}$, i.e. the length of $\mathcal{P}_{\text{REF}}$ and $N_M, N_{M-1}, ..., N_2, N_1$ are the number of phoneme candidates for substitution for each phoneme unit in $\mathcal{P}_{\text{REF}}$, including the reference phoneme itself. When M increases, assuming $N_m = 4$ for all phonemes p_m, where $m \in [1, M]$, then based on Equation (4.4), the number of candidate pronunciations $X = \prod_{m=1}^{M} N_m = 4^M$ will increase exponentially, which will significantly increase the computational cost. Hence, it is necessary and important to constrain the phoneme search radius (threshold) when the length of $\mathcal{P}_{\text{REF}}$ M is larger than a certain threshold $M_{\max}$. In this project, $M_{\max} = 6$ and for each phoneme p_n we constrain the search radius r_n to $r_n^{'}$, where

$$r_m^{'} = \begin{cases} r_m & M \leq M_{\max} \\ (\dfrac{M_{\max} - 1}{M - 1}) r_m & M > M_{\max} \end{cases} \tag{4.21}$$

For example, to generate candidate pronunciations for the word `desjardins` in the name `marine desjardins`, the number of candidate pronunciations and the corresponding computational cost will be significantly different, when applied with or without radius search constraint. This is illustrated in Table 4.14.

Table 4.14 Example of generating candidate pronunciation with and without search radius constraints

Condition	**No. of candidate phonemes (*M*=*10*)**										***X***	**Cost**
	d	eh	s	zh	aa	r	d	iy	n	z		
unconstrained	4	5	6	6	7	3	4	5	3	5	4,536,000	5 hour 35 minutes
constrained	2	4	5	4	5	3	2	4	3	4	230,400	17 minutes

Figure 4.7 shows the difference of the number of candidate pronunciations X when the dynamic threshold is enabled (the blue curve with "+") and disabled (the red curve with "$*$") in the pilot database. There are 174 out of 916 words in this database (only

19%) with long enough reference pronunciations to trigger the dynamic threshold constraint. With this technique, the average number of candidate pronunciations is significantly reduced from 20204 ($\bar{X}_2$) to 11941 ($\bar{X}_1$). It improved the efficiency with just a little compromise in accuracy.

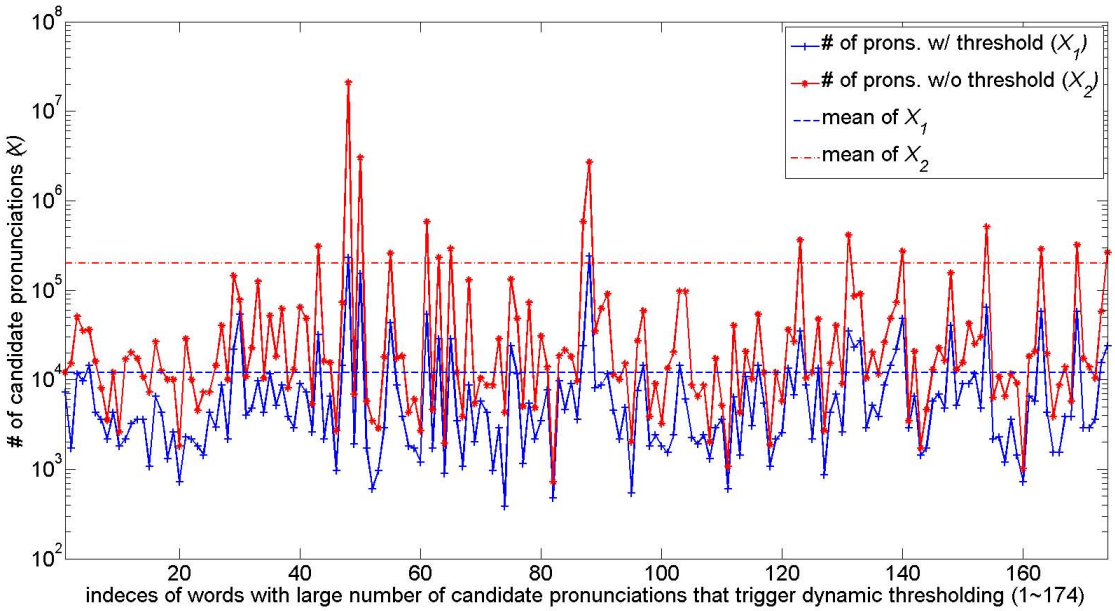


Fig. 4.7. Comparison of the number of candidate pronunciations with or without dynamic threshold in pilot database

As it is discussed in the section on hierarchical pronunciation learning, the phoneme units in the candidate pronunciations are determined one by one. Given M is the length of the reference pronunciation $\mathcal{P}_{\text{REF}}$, and N_i where $i \in [1, m]$ is the number of phoneme candidates including the reference phoneme itself for the mth phoneme unit, then $L_m = \prod_{i=1}^{m} N_i$ is the number of pronunciations processed when determining the mth phoneme, and $L = \sum_{m=1}^{M} L_m$ is the total number of pronunciations processed when learning the pronunciation for one word.

In the example given in Figure 4.3, the three phoneme unit in the word `paine` is determined in the order $n_3 \rightarrow n_2 \rightarrow n_1$ which is the natural order of their number of phoneme candidates $N_3 \rightarrow N_2 \rightarrow N_1$. However, if the phoneme units are determined in descending order of their number of phoneme candidates, i.e. processing phoneme

determination in the order of $n_2 \rightarrow n_3 \rightarrow n_1$, since $N_2 \geq N_3 \geq N_1$, then, the total number of processed pronunciations L will be minimized. Compared to the natural order, the total number of pronunciations processed is $L_{\text{descend}} = 22$ (Figure 4.8) rather than $L_{\text{natural}} = 26$ (Figure 4.3). Conversely, if the phoneme units are determined in ascending order, i.e., $n_1 \rightarrow n_3 \rightarrow n_2$, since $N_1 \leq N_3 \leq N_2$, the total number of processed pronunciation L will be minimized. In the example demonstrated in Figure 4.9, the total number of processed pronunciations $L_{\text{ascend}} = 28$. Thus, in this example, $L_{\text{descend}} = 22$, $L_{\text{natural}} = 22$ and $L_{\text{ascend}} = 28$, which satisfy

$$L_{\text{descend}} \leq L_{\text{natural}} \leq L_{\text{ascend}}. \tag{4.22}$$

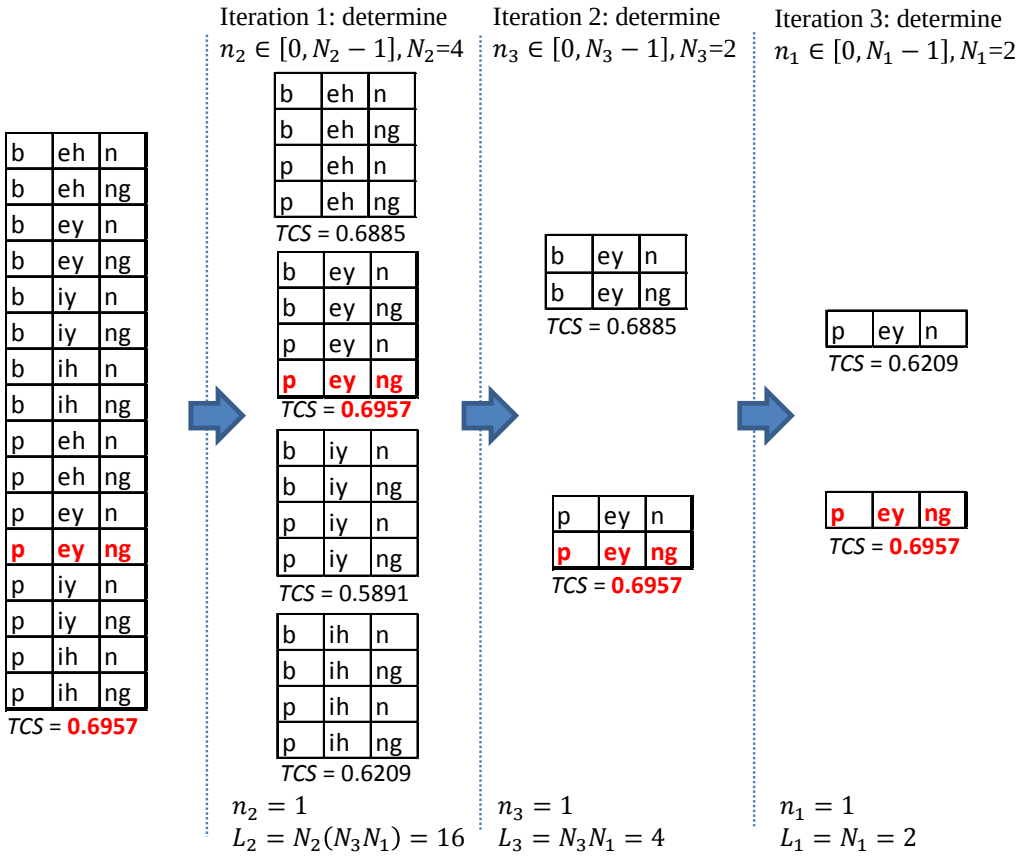


Fig. 4.8. Hierarchical pronunciation learning for the word `paine` with "descending" order of phoneme determination

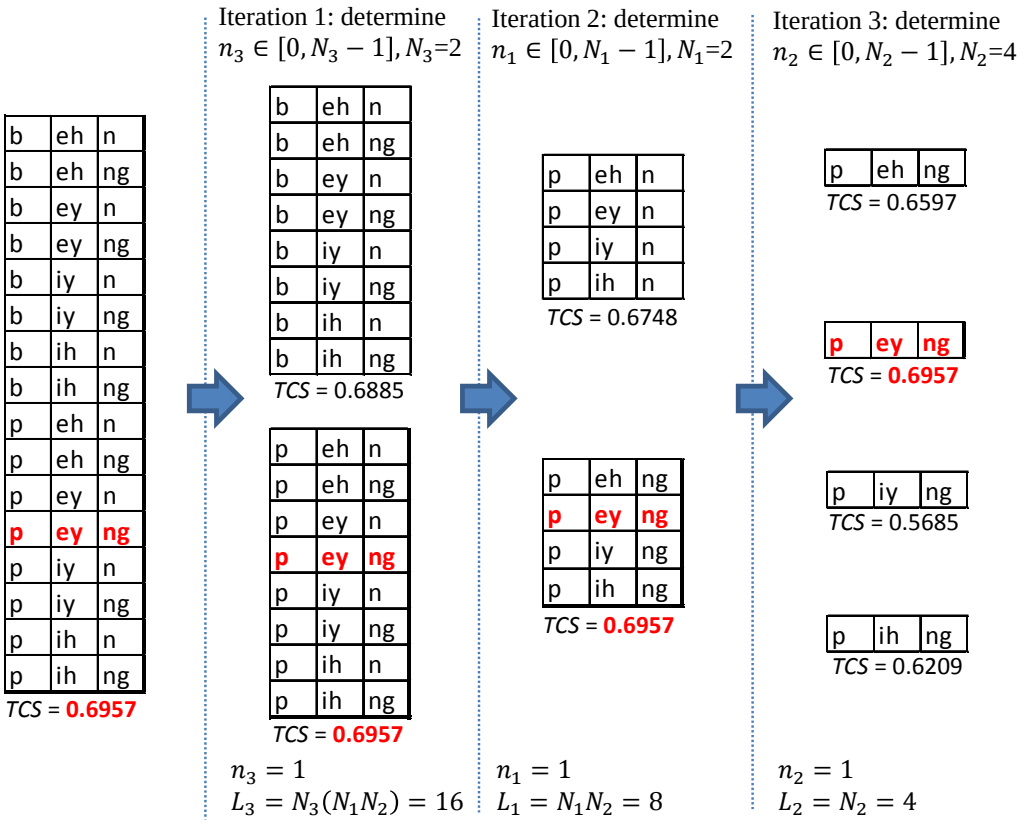


Fig. 4.9. Hierarchical pronunciation learning for word `paine` with "ascending" order of phoneme determination

In the following, Equation (4.22) will be proved to hold for any arbitrary pronunciation in general. Let $\{N_M, N_{M-1}, ..., N_2, N_1\}$ be the numbers of phoneme candidates for all phoneme units in $\mathcal{P}_{\text{REF}}$ listed in natural order. We re-order them in descending order $\{N'_M, N'_{M-1}, ..., N'_2, N'_1\}$ where $N'_M \geq N'_{M-1} \geq, ..., \geq N'_2 \geq N'_1$, based on the definition of the number of pronunciations processed when determining the mth phoneme L_m in Equation (4.11) and the total number of pronunciations processed L in Equation (4.12),

$$L_{\text{descend}} = \underbrace{\prod_{i=1}^{M} N'_i}_{\text{term 1}} + \underbrace{\prod_{i=1}^{M-1} N'_i}_{\text{term 2}} + \cdots + \underbrace{\prod_{i=1}^{1} N'_i}_{\text{term } M} = \sum_{m=1}^{M} (\prod_{i=1}^{m} N'_i) \tag{4.23}$$

and

$$L_{\text{ascend}} = \underbrace{\prod_{i=1}^{M} N_i^{'}}_{\text{term 1}} + \underbrace{\prod_{i=2}^{M} N_i^{'}}_{\text{term 2}} + \cdots + \underbrace{\prod_{i=M}^{M} N_i^{'}}_{\text{term } M} = \sum_{m=1}^{M} (\prod_{i=m}^{M} N_i^{'}). \tag{4.24}$$

Since $N_M^{'} \geq N_{M-1}^{'} \geq, ..., \geq N_2^{'} \geq N_1^{'}$, the terms in Equation (4.23) are always no greater than the corresponding terms in Equation (4.24) from term two to term M, i.e.

$$\prod_{i=m}^{M-m+1} N_i^{'} \leq \prod_{i=m}^{M} N_i^{'}, \quad m \in [2, M] \tag{4.25}$$

Thus,

$$L_{\text{descend}} \leq L_{\text{ascend}} \tag{4.26}$$

and for the same reason, L_{natural} will be in between L_{descend} and L_{ascend}, hence (Equation 4.22) is proved.

Based on Equation (4.13), the total time cost T in hierarchical pronunciation learning is mainly composed of the recognizer running cost T_{RUN} and the pronunciation processing cost T_{RPON}. The later term can be minimized or maximized based on the phoneme determination order, while the former term does not change whatever phoneme determination ordering is used. Figure 4.10 shows the total cost T of pronunciation learning on the pilot database, when T_{RPON} is minimized or maximized when processing phoneme determination by descending or ascending order of N_m. As it is shown in the figure, the advantages of descending ordering is more apparently when the word/name instances contain larger numbers of candidate pronunciations.

Multi-scale Testing and Results

The performance of pronunciation learning with various grammar scale has been shown in Table 4.15. As the number indicates, when the grammar size grows, pronunciation learning alone may even reduce the overall recognition performance, and pronunciation pruning after learning become necessary and more and more important.

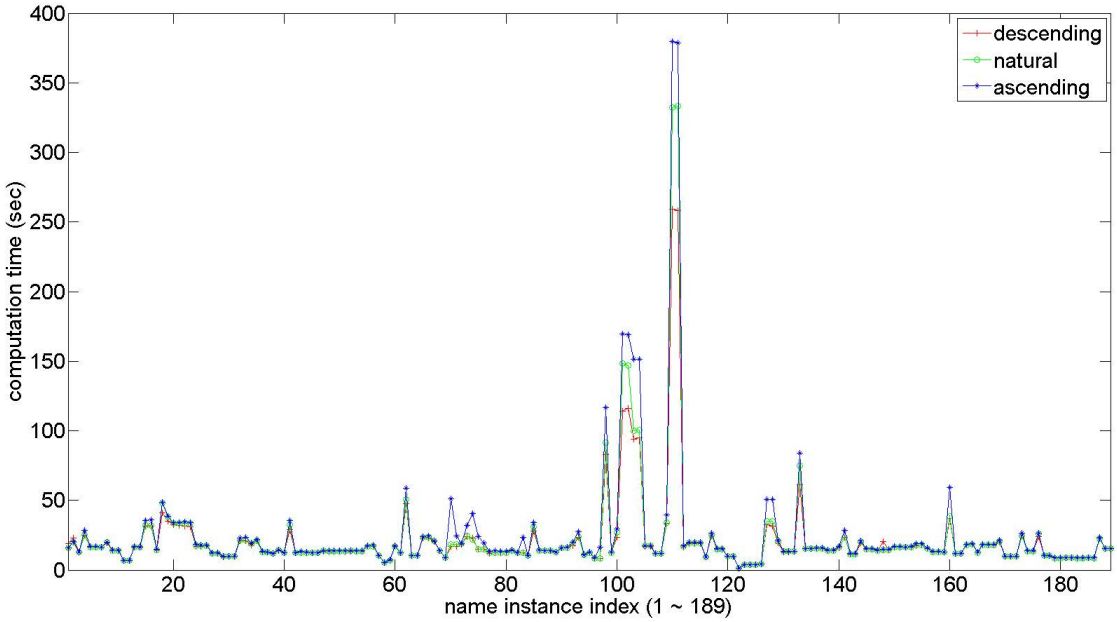


Fig. 4.10. Comparison of computational cost with different order in phoneme determination

Table 4.15 Performance of name recognition with different grammar scales

Database	Baseline	Learn	Learn + Prune
Pilot (610)	92.3%	95.2%	97.7%
Phase1-1000	91.5%	92.3%	94.9%
Phase2-3000	87.3%	86.0%	93.4%
Phase2-5000	84.9%	83.8%	92.2%

4.1.7 Summary and Future Work

The pronunciation dictionary is in fact a list of words and their corresponding pronunciations in terms of phoneme sequences, i.e., a list of word-phoneme sequence pairs.

Learning weighted pronunciations

The pronunciations of names are learned at the word level. Because of the difference in popularity of different words, the learned pronunciations of these words should

be treated differently by their "weights", so the names with more popular words get better chances to be correctly recognized, which eventually should improve the overall performance of the name recognition.

Extendability of Learned Pronunciations

Pronunciation learned from one database can benefit recognition of another database. Adding dictionary entries learned from one database, potentially will benefit recognizing names from another database, given these two name databases share some words. For example, adding pronunciations learned from the ININ data collection database (13875 unique names and 38806 name instances), to the lexicon of another much smaller database, company directory database (586 unique names and 1940 name instances), improves the recognition accuracy on the smaller database.

Learning Pronunciation Rules instead of Pronunciations Themselves

Using the STP interpreter to predict the rules of alternating original pronunciations to new pronunciations might lead to improvement. With the input of the original pronunciation and the learned pronunciation of words, the STP interpreter can learn the rules for changing the original pronunciation being changed to the newly learned pronunciation. Applying these rules to new words not covered in the pronunciation learning, we may generate better pronunciation for these new words. It allows the original pronunciations to automatically evolved to better pronunciations without pronunciation learning and makes the learned pronunciation from one database beneficial for other arbitrary database, even when they do not share any words.

4.2 Accent Classification using Acoustic and Phonetic Information

Mispronunciation detection addressed in this section, is aimed at improving speech recognition for speakers with accents. Speech recognition in this context is becoming

increasing more important as businesses become more international. Handling calls with accents is a major challenge for companies specializing in speech recognition support services. The particular problem we address in this thesis is identifying the accent of the caller from a short segment of speech. In this paradigm, the caller is speaking English. The algorithm analyzes the callers speech with respect to accent. After detecting the customer's accent, an accent-adapted speech recognition engine can be employed to recognize accented speech better.

The conversational nature of calls makes it especially hard from a speech recognition standpoint. North American English poses challenging problems because of the diversity in the English speaking population. One of the main contributors to poor accuracy in speech recognition is accentual variation in pronunciation of words in the vocabulary. In theory, deviations from canonical forms could be handled by hiring an expert phonetician or linguist to construct an expansive phonetic dictionary. This process, however, would be extremely time consuming and could result in a dictionary that is unwieldy.

Compared with mispronunciation detection for language learning, here instead of pointing out the specific part of mispronounced phones or words during speech, it only requires a general decision of whether the speaker is accented or not, based on the whole recorded speech. If the speaker has an accent, we need to determine the accents he/she has, like Spanish accent, Chinese accent, or Indian accent. This problem is challenging for a couple of reasons.

First, we only have a short amount of time to determine the speaker's accent, which means data collected during this short period of time may not be enough to detect accents with high accuracy. Second, there is no transcription that accompanies the customer calls, which means the customer's accents have to be determined based on text-independent speech. This is totally different from the case in language learning, where the detection system knows exactly what the speaker will pronounce.

In this thesis, two types of accent-adapted speech recognition engines are designed and tested. We have access to ININ's customer call data. The current state of the art

is about 35% accuracy for classifying 23 accents in American English. Since this application of mispronunciation detection is based on data of large vocabulary continuous speech, before discussing the proposed methods for developing an accent-adapted speech recognition engine, a general introduction of Large Vocabulary Continuous Speech Recognition (LVCSR) is provided next.

4.2.1 Continuous Speech Recognition Using Phone Recognizer

Generally speaking, LVCSR system are comprised of an ASR engine and a Language Model (LM). The ASR engine is a statistical model usually based on HMMs, and LM is a statistical language model that assigns a sequence of speech units by means of a probability distribution. LVCSR is commonly performed at the phone level, so both HMM and LM will be trained with phone-level data with transcription.

The following word-level example will briefly explain the concept of LM and then compares it with an HMM. Given a sentence “you read my thesis", the probability of this sentence is equal to

$$\begin{aligned} P(you, read, my, thesis) &\approx P(you|<s>)P(read|you)P(my|read)\cdots \\ &\quad P(thesis|my)P(</s>|thesis), \end{aligned} \tag{4.27}$$

in a bigram ($n = 2$) LM, whereas in a trigram ($n = 3$) LM, it approximated as

$$\begin{aligned} P(you, read, my, thesis) &\approx P(you|<s>,<s>)P(read|<s>, you)\cdots \\ &\quad P(my|you, read)P(thesis|read, my)\cdots \\ &\quad P(</s>|my, thesis), \end{aligned} \tag{4.28}$$

where $< s >$ and $/s$ are the beginning and ending marks of the sentence. An n-gram LM contains an $(n-1)$-gram LM and a list of probability $P(w_i|w_{i-(n-1)}, \cdots, w_{i-1})$ of observing sequence $(w_1, w_2, \ldots, w_m)$ which can be approximated as

$$\begin{aligned} P(w_1, \ldots, w_m) &= \prod_{i=1}^{m} P(w_i|w_1, \ldots, w_{i-1}) \\ &\approx \prod_{i=1}^{m} P(w_i|w_{i-(n-1)}, \cdots, w_{i-1}), \end{aligned} \tag{4.29}$$

where m is the length of the sequence being examined. $P(w_i|w_{i-(n-1)}, \cdots, w_{i-1})$ must be larger than the predefined cut-off threshold θ_n to be counted in the list. If $P(w_i|w_{i-(n-1)}, \cdots, w_{i-1})$ appears in the testing data which cannot be found in LM, its probability will be computed using $P(w_i|w_{i-(n-2)}, \cdots, w_{i-1})$ and $P(w_{i-(n-2)}|w_{i-(n-1)})$ with some back-off strategies, which will cause significant probability drop.

Compared with the HMM, which handles the utterance at the acoustic level (i.e. how it sounds), the LM handles the utterance at the context or language level. The final recognition result of the test utterance will be determined by both the HMM and the LM.

The commonly used open source ASR engines include the Hidden Markov Model Toolkit (HTK) [49], Sphinx [50], and Julius [51]. The most popular LM generation tools are the Sphinx Knowledge Base Tool from Carnegie Mellon University (CMU) and Cambridge University [52] and SRILM from Stanford Research Institute (SRI) [53].

After clarifying the task, goal and potential challenges in the application of mispronunciation detection for speech recognition adaptation, and briefly introducing the baseline methodology of VLCSR, in the following Section 4.2.2, two proposed methods to design accent-adapted phone recognizer with text-dependent speech are outlined. In Section 4.2.3, the performance of the proposed accent-adapted phone recognizer with text-independent speech will be measured. The framework of phone recognizer system design and performance test is illustrated in Figure 4.11.

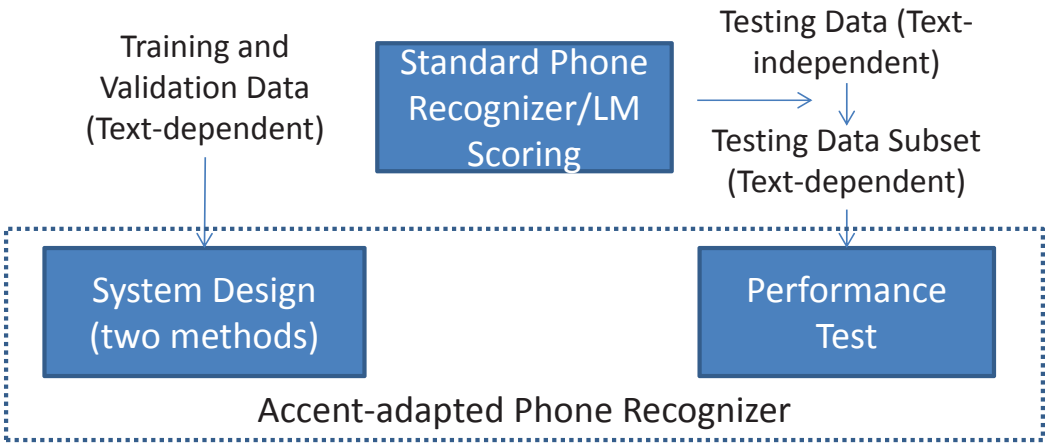


Fig. 4.11. Framework of accent-adapted phone recognizer

4.2.2 Design of Accent-adapted Phone Recognizer with Text-dependent Speech

In this section, two methods are proposed to adapt standard phone recognizers to multiple accent-adapted phone recognizers for recognizing multiple foreign accents where callers are speaking American English. The data used in this part is native and non-native continuous speech with transcriptions.

Accent-adapted Language Models Based on Phone Mapping

The simple idea behind this proposed method is to construct a list of phone mapping pairs, which map the correct phones in the target language (American English) to certain type of accented phones in the speaker's native language, such as Chinese, Spanish, Hindustani, etc.

Consider the case of native Chinese speakers speaking English. Some of the frequent phone mapping pairs are:

- /ɵ/ → /s/, ex: pronounce “think" like “sink"
- /ə:/ → /ɔ/, ex: pronounce “work" like “walk"
- /æ/ → /e/, ex: pronounce “back" like “beck"

Most of the phone mappings do not come along with the word correspondences, and a completed list of phone mappings is not easy to obtain by humans even for a linguistic expert.

Fortunately, given two LMs trained from native and non-native corpuses respectively, native LM can be adapted towards the non-native one by 1) finding out these phone mapping pairs sorted by frequency and 2) rebalancing the probabilities of those phone pairs up to the n-gram model in the native LM. A simple example of speech recognition on non-native speech is shown in Table 4.16, using the Wall Street Journal (WSJ) corpus as the non-native corpus and the Fisher corpus as the native one. WSJ corpus is recorded with regional New York accents of English, while the Fisher corpus is more general and represent the entire North American English population with various accents. This example illustrates that potential phone mapping methods can be used to adjust the general Fisher LM towards the WSJ LM to accommodate New York accents. The speech recognition is conducted using the Julius speech recognition engine with a universal phonetic alphabet of English called "Arpabet" [47].

Table 4.16 Recognition of non-native speech using native (Fisher) corpus and non-native (WSJ) corpus

Utterance	I		did			a		lot				of		soul			searching								
Reference	ay		d	ih		d	ah	l	aa		t	ah		v	s	ow	l	s	er	ch	ih		ng	
WSJ	ay		d	ih		d	ah	l	aa		t	ah		v	s	ow	l	s	er	ch	ih		ng	
Fisher	ay	*	**	d	**	*	**	UW				AO	R	s	**	AH		s	er	ch	ih		ng		

*: missing phone; letter capitalized: mis-recognized phone

The 6-gram LM score for each 6-frame time period is computed moving one frame after another with both LMs. Two curves with the 6-gram LM log likelihood are illustrated in Figure 4.12. By observing the changes in both curves, the most significant drop occurs in window 12 (/s ow l s er ch/). Missing /ow/ and mis-recognized /l/ as /ah/ are the two major defects contributing to this drop. This means two

potential native-to-non-native phone pairs are 1) /ow/ vs. the phone with highest log likelihood in the native LM; 2) /l/ vs. /ah/.

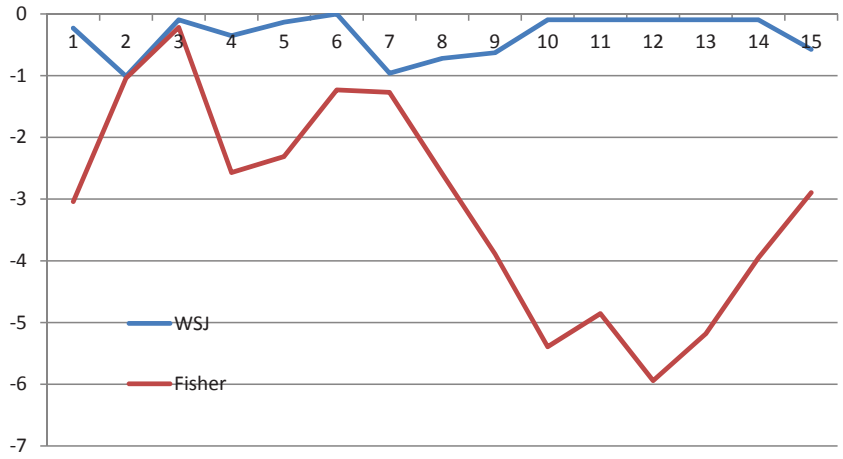


Fig. 4.12. Log likelihoods based on native (Fisher) and non-native (WSJ) LMs

Accent-adapted Features Based on Vowel Representation

The previous method of designing accent-adapted phone recognizers in Section **??** is based on language model adaptation with phone mapping information. In this section, another method based on accent-adapted vowel representation in feature space will be discussed. This proposed method is inspired by the work from Minematsu et al. [54] and Suzuki et al. [55], where they measured the overall structure of the speaker's phonetic space.

For each type of accented version of the targeting language, as well as the standard one, it is assumed that the MFCC of the five fundamental vowels are located relatively constantly in the feature space. In Figure 4.13, the first two dimensions of MFCC are taken to illustrate the position of five accents in accented and non-accented languages [54]. The center in each pentagon is the weighted average of five vowels based on their positions in feature space and frequency of appearance in the corpus. By matching the center of the pentagon of standard and accented language into the overlapped pentagon in the bottom of Figure 4.13, the Bhattacharyya distances [56] between

each pair of corresponding vowels and their angles can be computed and stored in a vector. This vector V_i represents the difference from the accented language L_i to the standard one L.

To classify the test speech into one of the accent categories $L_1, L_2, \ldots, L_N$, where N is the number of accent categories, the difference from V_j to $V_i, i \in [1, N]$ and V (category of standard language) are computed, compared and classified to the nearest category of accent.

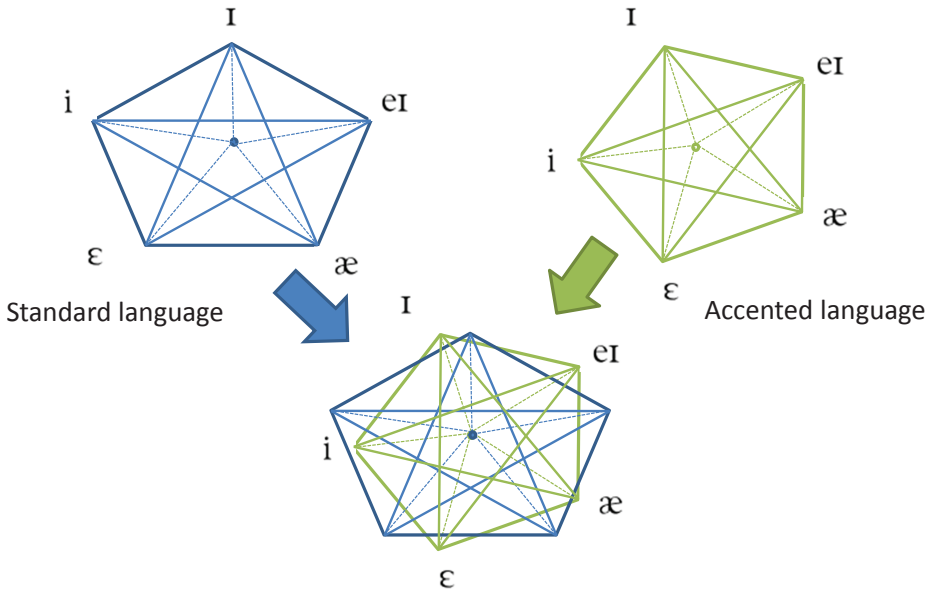


Fig. 4.13. Comparison of 5 vowels locations in standard and accented language

4.2.3 Performance Measurement of Accent-adapted Phone Recognizer with Text-independent Speech

As illustrated in Figure 4.11, the performance of the proposed accent-adapted phone recognizer will be finally measured with text-independent data, which is the case for customer calls. These test data are based on topic-oriented conversations and hence will have different LMs and variations of accents, compared with the training data.

Test data with transcription is necessary to test the performance of accent detection and classification. The following procedure will be used to convert text-independent data from customer calls to text-dependent speech, and then to test the performance of the proposed accent-adapted phone recognizers:

1. Perform speech recognition using standard LM on the text-independent call data;
2. Select a subset of recognized speech with a certain level of confidence based on the n-gram log likelihood of recognized speech, such as log likelihood (n-gram)$\geq \theta$, where $n \geq 5$ and θ is the predefined threshold to ensure the confidence of accuracy;
    - this step converts text-independent speech to a subset of text-dependent speech with a certain confidence level;
    - appropriate threshold will be determined with text-dependent training data.
3. Perform accent detection and classification using two types of proposed accent-adapted phone recognizer. The results will be compared with human-labeled customer data.

4.2.4 Data Preparation in Accent Classification

The database used for developing the accent classification system in this thesis is Foreign Accented English (FAE) corpus with catalog number LDC2007S08, purchased from Linguistic Data Consortium (LDC). It was originally collected by the Center of Speech & Language Understanding (CSLU) at Oregon Health & Science University (OHSU). It contains 4925 accented English speech sentences about 20 seconds long each, from speakers with 23 types of language origins, which are listed in Table 4.17. These accented speech recordings are grouped into seven clusters based on the relationship of the accents by a computational linguist from ININ. Dur.$_1$ and Dur.

Table 4.17 Summary of the Foreign Accented English (FAE) corpus

ID	Group	Accent	Abbr.	No. of speech	Prop.(%)	Dur.$_1$	Dur.$_2$	Comp. rate (%)
1	1	Arabic	AR	112	2.27	0:34:32	0:29:11	84.5
2	4	Brazillian Portuguese	BP	459	9.32	2:34:24	2:09:58	84.2
3	3	Cantonese	CA	261	5.3	1:17:34	1:05:59	85.1
4	5	Czech	CZ	102	2.07	0:33:25	0:28:31	85.3
5	6	Farsi	FA	261	5.3	1:18:56	1:03:21	80.3
6	2	French	FR	284	5.77	1:31:05	1:18:44	86.4
7	6	German	GE	325	6.6	1:36:04	1:22:18	85.7
8	7	Hindi	HI	348	7.07	1:56:10	1:36:31	83.1
9	5	Hungarian	HU	276	5.6	1:27:20	1:13:33	84.2
10	2	Indonesian	IN	96	1.95	0:31:19	0:25:50	82.5
11	4	Italian	IT	213	4.32	1:04:07	0:53:30	83.5
12	3	Japanese	JA	194	3.94	0:56:05	0:47:33	84.8
13	3	Korean	KO	169	3.43	0:53:35	0:44:54	83.8
14	3	Mandarin	MA	282	5.73	1:30:37	1:16:06	84.0
15	2	Malay	MY	56	1.14	0:17:21	0:14:37	84.2
16	5	Polish	PO	143	2.9	0:47:04	0:40:01	85.0
17	4	Iberian Portuguese	PP	66	1.34	0:21:08	0:16:46	79.3
18	5	Russian	RU	236	4.79	1:11:13	0:59:54	84.1
16	6	Swedish	SD	203	4.12	1:07:37	0:58:14	86.1
20	4	Spanish	SP	308	6.25	1:05:19	0:53:45	82.3
21	2	Swahili	SW	71	1.44	0:21:34	0:18:16	84.7
22	7	Tamil	TA	326	6.62	1:06:29	0:54:31	82.0
23	3	Vietnamese	VI	134	2.72	0:27:12	0:21:26	78.8

$_2$ denote the total duration of the speech per accent type before and after the pre-processing of speech with silence removal. The technique used for silence removal includes the measurement of short-time energy rate and spectral centroids described in [57]. After removing the silence, the duration of speech in each type of accent is reduced to the range between 78.8% and 85.7%, of the original duration, as indicated by the compression rate.

Figure 4.14 is an example to demonstrate silence removal using short-time energy rate and spectral centroids on audio file FAR00042.wav in the speech database with Arabic accents. The portion of speech is considered to be silence when either the smoothed short-time energy rate and the smoothed spectral centroids are below

certain thresholds. The short-time energy rate is used to remove the environmental noise. While spectral centroids, can be used to remove non-speech noise, such as coughing, due to its lower energy in the spectrum, relative to that of regular human speech.

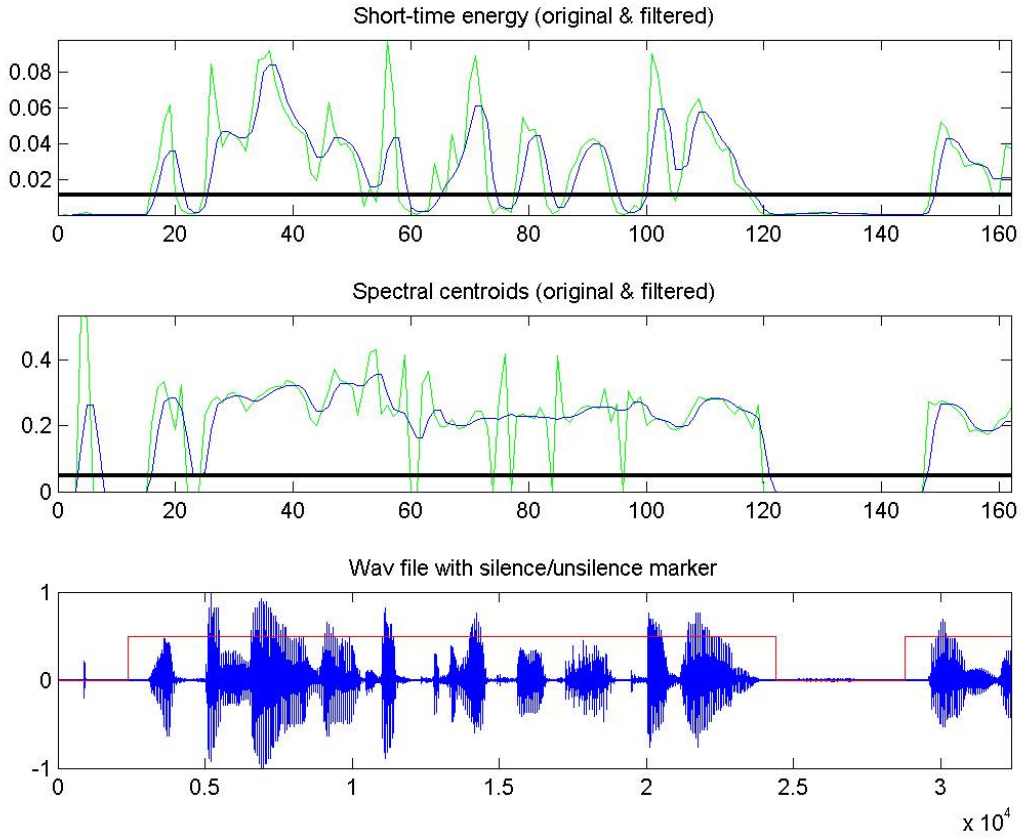


Fig. 4.14. Example of silence removal using short-time energy rate and spectral centroids (FAR00042.wav in FAE corpus)

4.2.5 Accent Classification based on Pure Acoustic Information

The accent classification based on acoustic information is implemented using a Gaussian Mixture Model (GMM) classifier with Perceptual Linear Predictive (PLP) features discriminatively optimized by Heteroscedastic Linear Discriminant Analysis (HLDA), similar to the methods described in [58]. HLDA is a generalization of Linear

Discriminant Analysis (LDA), which allows features to have different variances in different feature dimensions.

Introduction of GMM

Motivated by the success in modeling attributes of speakers using Gaussian Mixture Models (GMMs) from [59], here we use GMMs to model the attributes of accents. Gaussian mixture density models the feature distribution of each accent as a weighted sum of multiple Gaussian distributions. For each feature vector $\mathbf{x}$ in the $M \times T$ feature set X, the probability of $\mathbf{x}$ can be formulated by the equation

$$p(\mathrm{x}|\lambda) = \sum_{i=1}^{N} p_i b_i(\mathbf{x}), \tag{4.30}$$

where

$$b_i(\mathbf{x}) = \frac{1}{(2\pi)^{M/2}|\Sigma_i|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i)\}. \tag{4.31}$$

N in Equation (4.30) is the number of mixture components, M in Equation (4.31) is the dimension of the feature vector $\mathbf{x}$, $b_i(\mathbf{x})$, $i = 1, ..., N$, are the component densities, p_i, $i = 1, ..., N$, are the mixture weights and $\lambda = \{p_i, \mu_i, \Sigma_i\}$, $i = 1, 2, ..., N$ is the collective representation of the parameters.

Given MFCC feature X $(M \times T)$ from accent type s, the Maximum Likelihood Estimation (MLE) is used to maximize the GMM likelihood, which can be written as

$$\lambda^* = \arg\max_{\lambda} p(\mathbf{X}|\lambda) = \arg\max_{\lambda} \prod_{t=1}^{T} p(\mathbf{x}_t|\lambda). \tag{4.32}$$

Since this expression is non-linear and direct maximization is difficult, the parameter set $\lambda = \{p, \mu, \Sigma\}$ is iteratively estimated using a special case of the Expectation-Maximization (EM) algorithm [60] and is summarized below:

$$\begin{aligned} \bar{p}_i &= \frac{1}{T}\sum_{t=1}^{T} p(i|\mathbf{x}_t, \lambda); \\ \bar{\mu}_i &= \frac{\sum_{t=1}^{T} p(i|\mathbf{x}_t, \lambda)\mathbf{x}_t}{\sum_{t=1}^{T} p(i|\mathbf{x}_t), \lambda)}; \\ \bar{\sigma}_i^2 &= \frac{\sum_{t=1}^{T} p(i|\mathbf{x}_t, \lambda)\mathbf{x}_t^2}{\sum_{t=1}^{T} p(i|\mathbf{x}_t, \lambda)} - \bar{\mu}_i^2, \end{aligned} \tag{4.33}$$

where $\bar{p}_i, \bar{\mu}_i, \bar{\sigma}_i^2$, $i = 1, ..., N$ are the mixture weights, means, and variances for the ith component; $p(i|\mathbf{x}_t, \lambda)$ is the *a posteriori* probability for the i-th component given by

$$p(i|\mathbf{x}_t, \lambda) = \frac{p_i b_i(\mathbf{x}_t)}{\sum_{k=1}^{M} p_k b_k(\mathbf{x}_t)} \ . \tag{4.34}$$

These estimates are based on the assumption of independence among feature dimension, so for each accent type s, the non-zero values of the covariance matrix are only on the diagonals. This algorithm guarantees a monotonic increase of the model's likelihood on each EM iteration.

After obtaining the GMM parameter set λ_s for speaker class $s \in [1, S]$, the GMM-based classifier, which maximize *a posteriori* probability for a feature sequence X, $(M \times T)$ can be formulated as:

$$\begin{aligned} \hat{S} &= \arg\max_{s\in[1,S]} \Pr(\lambda_s|X) \\ &= \arg\max_{s\in[1,S]} \frac{p(X|\lambda_s)\lambda_s}{p(X)} \\ &\propto \arg\max_{s\in[1,S]} p(X|\lambda_s) \\ &\propto \arg\max_{s\in[1,S]} \sum_{t=1}^{T} \log p(\mathbf{x}_t|\lambda_s). \end{aligned} \tag{4.35}$$

The first equation is due to Bayes' rule. The first proportion is assuming $\Pr(\lambda_s) = 1/S$ and $p(X)$ is the same for all speaker models. The second proportion uses logarithm and independence between input samples $\mathbf{x}_t$, $t \in [1, T]$.

Introduction of LDA

Linear Discriminant Analysis (LDA) is a data dimension reduction technique that maps data into a subspace while maximizing the discriminative information. For the discussion of LDA, assume there are $T = \sum_{s=1}^{S} T_s$ number of M-dimensional data vectors $\boldsymbol{x}_t$ in S classes, where T_s is the number of vectors in class $s \in [1, S]$. Let the global mean $\boldsymbol{\Phi}$ over all classes and the local mean $\boldsymbol{\Phi}_s$ for each class s be

$$\boldsymbol{\Phi} = \frac{1}{T} \sum_{t}^{T} \boldsymbol{x}_t \tag{4.36}$$

and

$$\boldsymbol{\Phi_s} = \frac{1}{T_s} \sum_{\boldsymbol{x}_t \in s} \boldsymbol{x}_t, \tag{4.37}$$

respectively. Then, we define between-class scatter S_B and within-class scatter S_W by

$$\begin{aligned} S_B &= \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\Phi})(\boldsymbol{x}_t - \boldsymbol{\Phi})^T \text{ or} \\ S_B &= \frac{1}{S} \sum_{s=1}^{S} (\boldsymbol{\Phi}_s - \boldsymbol{\Phi})(\boldsymbol{\Phi}_s - \boldsymbol{\Phi})^T, \end{aligned} \tag{4.38}$$

and

$$\begin{aligned} S_W &= \frac{1}{S} \sum_{s=1}^{S} \sum_{\boldsymbol{x}_t \in s} (\boldsymbol{x}_t - \boldsymbol{\Phi}_s)(\boldsymbol{x}_t - \boldsymbol{\Phi}_s)^T \text{ or} \\ S_W &= \frac{1}{S} \sum_{s=1}^{S} \frac{1}{T_s} \sum_{\boldsymbol{x}_t \in s} (\boldsymbol{x}_t - \boldsymbol{\Phi}_s)(\boldsymbol{x}_t - \boldsymbol{\Phi}_s)^T. \end{aligned} \tag{4.39}$$

The first definitions of Equation (4.38) and Equation (4.39) consider the class weights, i.e. the sizes of each class s, while the second does not. The first definitions are used in this work for the consistency with the LDA definition used in Kumar's HLDA work [61]. However, the second definitions [62] of both formulas are also provided for completeness.

If we choose $\mathbf{w}$ from the underlying space W, then $\mathbf{w}^T S_B \mathbf{w}$ and $\mathbf{w}^T S_W \mathbf{w}$ are the projections of S_B and S_W onto the direction $\mathbf{w}$. Searching the directions $\mathbf{w}$ for the best class discrimination is equivalent to maximizing the ratio of $(\mathbf{w}^T S_B \mathbf{w})/(\mathbf{w}^T S_W \mathbf{w})$ subject to $\mathbf{w}^T S_W \mathbf{w} = 1$. The latter is called the Fisher Discriminant Function and can be converted to

$$S_B \mathbf{w} = \lambda S_W \mathbf{w}, \text{ then } S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w} \tag{4.40}$$

by Lagrange multipliers and solved by eigen-decomposition of $S_W^{-1} S_B$.

By selecting eigenvectors associated with the most significant m eigenvalues of $S_W^{-1} S_B$, one can map the original M-dimensional data into a m-dimensional subspace for discriminative feature reduction.

Introduction of HLDA

LDA is derived with the assumption that features in various dimensions have the same variance, which may not be the case in the real problem. For example, consider two classes of data with the Gaussian distributions shown in Figure 4.15. They have the same variance and slightly different mean in one direction, while same mean and significantly different variance in the other distribution. LDA will project the data to the first direction, since it maximizes the ratio of between-class scatter S_B and within-class scatter S_W. However, the other direction will lead to the best discriminant information in this case.

This work uses Kumar's method [61] to eliminate this assumption and generalize LDA to HLDA using Maximum Likelihood Estimation (MLE) on Gaussian distri-

butions. The improved version of Kumar's HLDA implementation in MATLAB, including GMM-HLDA classifier, is attached in Appendix B.

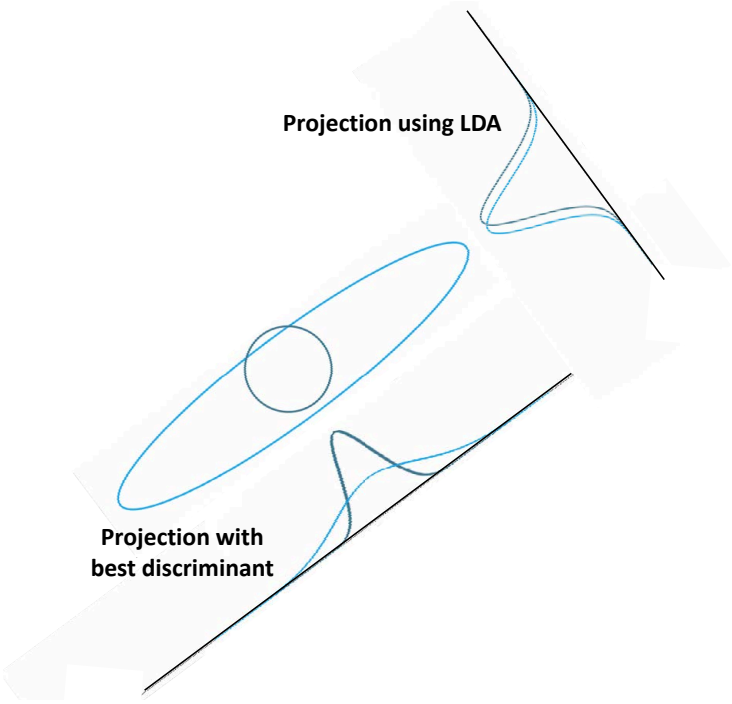


Fig. 4.15. Illustration of how LDA fails with two Gaussian distributions

Implementation of Accent Classification based on Pure Acoustic Information

The diagram of accent classification based on pure acoustic information is demonstrated in Figure 4.16. Data from seven major types of accents, including AR, BP, FR, GE, HI, MA and RU are used in implementation. They are divided into training (70%), development (15%) and testing (15%) based on the numbers of recordings. Features of 39-dimensional PLPs with Mean and Variance Normalization (MVN) are extracted and further improved using HLDA with context-size 1 and reduced dimension 20. The context-size factor is used to duplicate features for potential performance improvement. For example, with context-size 1, the original feature frame is elongated with the concatenation from its 1 left frame and 1 right frame. Both the GMM classi-

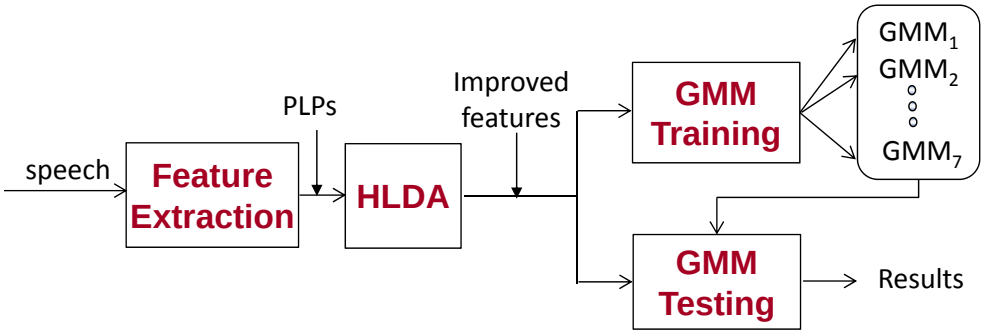


Fig. 4.16. Diagram of accent classification based on pure acoustic information

fier and the improved GMM-HLDA classifier trained with features of various types of accents represent accents with GMMs of 256 Gaussian Mixtures. These parameters, including GMM order, feature dimension in PLP and HLDA, and context-size are tuned using development data set. The performance with the testing data set achieve 40% and 46% accuracies using GMM classifier and GMM-HLDA classifier.

4.2.6 Accent Classification based on Acoustic and Phonetic Information

In Section 4.2.2, two methods of accent classification using phonetic information are proposed. They are based on patterns of phoneme mapping and shifts of vowel representation respectively. Currently, a modified version of the second method is implemented. Instead of directly measuring the shifting of vowels, the same vowel of various types of accents are trained as GMMs separately. Instead of using fundamental five vowels, this method uses all fifteen vowels in Arpabet listed in Table 4.18, and

Table 4.18 Vowels in Arpabet

aa	ae	ah	ao	aw	ay	eh	er	ey	ih	iy	ow	oy	uh	uw
f**a**ther	f**a**st	s**u**n	h**o**t	h**ow**	m**y**	r**e**d	b**ir**d	s**ay**	b**i**g	m**ee**t	sh**ow**	b**oy**	b**oo**k	f**oo**d

rank them based on their performance in recognizing accents. Given T types of accents, a subset of vowels S_t can be found experimentally for tth accents and form

the GMM classifier as the combination of GMM classifiers of all vowels in subset S_t, which can be formulated as:

$$\text{GMM}_t = \sum_{i \in S_t} \text{GMM}(t, v_i), \tag{4.41}$$

where v_i is the ith vowels in vowel subset S_t of tth accent.

Adding this additional layer on the GMM classifier is critical to find the vowel sets which preserve the accents and shown to improve on classifying accents. However, it requires recognizing these vowels in the front end. During training and development, the phoneme alignment based on the ININ phoneme recognizer is used to extract the vowels. During testing, a subset of recognized vowels with certain level of confidence are selected after phoneme recognition, based on the proposed method in Section 4.2.3.

Dictionary Preparation and Phoneme Alignment for FAE Corpus

Before extracting vowels using phoneme alignment, it is necessary to obtain the transcription of FAE corpus which is originally absent in the LDC's release. People in ININ helped to partially transcribe the accented speech from major 13 out of 23 accents, including AR, BP, FA, FR, GE, HI, IT, KO, MA, RU, SP, TA, VI. Among them, data from AR, BP, FR, GE, HI, MA and RU are used in this work. With the output of dictionary preparation, such as dictionary file, word-level transcription of accented speech and data list file, phoneme alignment for vowel extraction is performed using HTK tools HVite.

Figure 4.17 demonstrates the process of dictionary preparation and phoneme alignment for FAE corpus. The dictionary file is a list of pairs of words and pronunciations in HTK format, which can be obtained through the process of word collection, word-to-pronunciation conversion with ININ Lexicon Tester and HTK dictionary file creation. In Phoneme alignment, the HTK configuration file, HMM model definition and tired list are all trained using Fisher corpus.

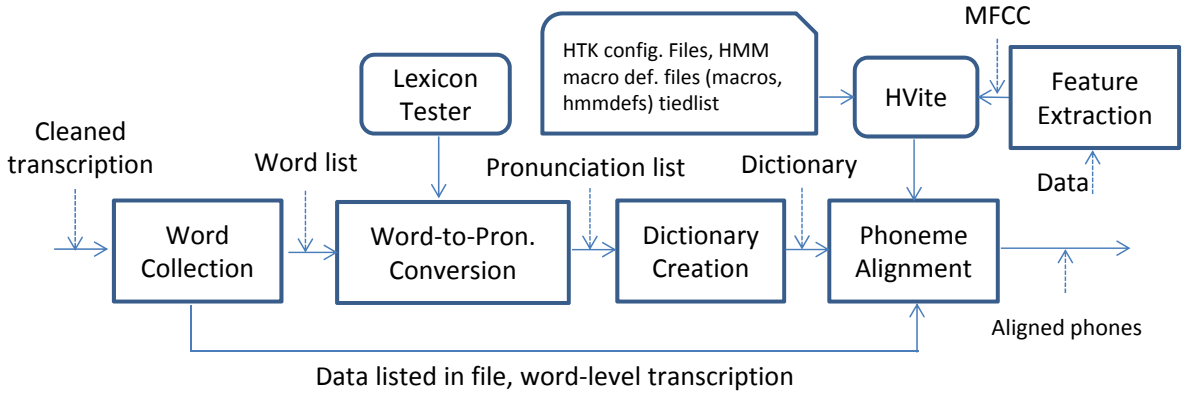


Fig. 4.17. Dictionary preparation and phoneme alignment for FAE corpus

Implementation of Accent Classification based on Both Acoustic and Phonetic Information

Here 39-dimensional MFCCs with MVNs are used in the implementation of accent classification. After training GMMs on seperated vowels, GMMs of 7 vowels out of 15 of each accent are selected to form the mixed GMM classifer for that accent. The overall classification accuracy is 51%, which gains 11% improvement from the GMM classifer trained with PLP features. Table 4.19 compares the performances of all three methods, including GMMs with PLPs, trained per accent; GMMs with HDLA-optimized PLPs, trained per accent; and GMMs with MFCCs, trained per accent and per vowel.

Table 4.19 7-way accent classification with acoustic and phonetic features

Method	$\mathbf{GMM}_{accent}$	$\mathbf{GMM}_{accent}$+**HLDA**	$\mathbf{GMM}_{vowel}$
Model	$\mathbf{GMM}^{256}_{accent}$	$\mathbf{GMM}^{256}_{accent}$	$\mathbf{GMM}^{256}_{vowel}$
Feature	$\mathbf{PLP}^{39}_{MVN}$	$\mathbf{PLP}^{39}_{MVN} + \mathbf{HLDA}^{20}_{C1}$	$\mathbf{MFCC}^{39}_{MVN}$
Accuracy	40%	46%	51%

4.2.7 Summary and Future Work

The work in accent classification shows the performance improvement with HLDA discriminative feature optimization and further selecting vowels over the whole accented speech for GMM classifer training and testing.

There are at least several areas that could be explored for further improvement:

1. Since the data for each accent is so limited (about 50 minutes per accent after pre-processing), a universal classifier based on Restricted Boltzmann Machine (RBM), instead of traditional GMMs for each accents, may be needed to explore [63]. RBM is trained using data of all accents, with capability to deviate with different accents.

2. Accent clustering based on certain distance measurements, such as Bhattacharyya distance [56] can also be used to pre-classify accents into several clusters of accents, which may potentially help narrow down the search and improve the classification accuracy.

3. In phoneme alignment and recognition based on tri-phone acoustic models, all phoneme units with the same mid-phone are treated the same currently for straight-forward implementation, the accent pattern may stay in the transition of phonemes, which can be taken care of later.

5. SUMMARY

In this thesis, we explore mispronunciation detection and classification in language learning and speech recognition adaptation. Though the specific methods we develop are different within these applications, the essential concept employed is the same, which is using the information from the speaker's own language to improve the statistical models for better detection of mispronunciations.

In the application of language learning, each phone in the entire phonetic alphabet is optimized to distinguish the correct pronunciation with all variations of that particular phone, using the optimal frequency scale in generating cepstral coefficients.

In the application of speech recognition adaptation, two types of adaptation mechanisms are developed. One is adapting the grammar-based speech recognition engine to variations in name pronunciation by learning additional but acceptable pronunciations of the same name. The other is developing accent classifier to classify accents, the classified accented speech can be used to adapt the speech recognition engine to better recognizing speech with particular accents.

Here is a list of the work that has been completed thus far:

- Word-adaptive frequency scales have been optimized to maximize the separation of two groups with correct and accented pronunciations respectively;
- PCA-based methods for mispronunciation detection and classification have been implemented and have achieved competitive performance when the size of training data is limited;
- A phone-mapping method for designing accent-adapted phone recognizer has been tested with Fisher (native) and WSJ (non-native) corpuses, and it shows promising results for improving speech recognition results for non-native speech.
- A hierarchical pronunciation learning algorithm is designed and developed to improve name recognition performance. This algorithm learns variation of name

pronunciations and avoids overlap of pronunciations of different names in name space through several effective and efficient pruning techniques;

- Several versions of accent classifiers were developed with discriminatively trained HLDA acoustic features. The phonetic information was also utilized to improve classification accuracy by extracting accent distinguishable vowels and build classifiers with separated vowels.

Here is a list of future work that could be explored later:

- Find text-adaptive frequency scales of the phone-level based on two criteria:

    1. maximize the separation of target phones and all other phones in the phonetic alphabet
    2. maximize the correlation with human scoring of the target phone with both native and non-native data

    The first criterion requires larger native data and the second criterion requires enough data in both native and non-native data to cover all variations of mispronunciation and phone-level scoring on both corpuses.
- Explore both methods to improve speech recognizer to recognize accented speech:

    1. develop phone mapping pairs sorted by contribution in speech recognition adaptation, and use these mappings to adjust standard LMs towards accent-adapted LMs accordingly;
    2. obtain the pentagon representation of 5 fundamental vowls in each accent-adapted feature space, and classify the test speech into one of these categories using Vector Quantization (VQ) principles;

- Convert text-independent speech to text-dependent speech and construct a subset of test data for accent-adapted phone recognizer performance tests:

    1. current proposed method is based on LM log likelihood threshold;
    2. modern data mining technique such as Latent Semantic Analysis (LSA) will be explored to obtain better text-dependent subsets with high confidence levels.

The potential contributions of this thesis include:

- New language learning systems that will allow users to receive automatic feedback when mispronunciation errors are made. Once detected, these errors can be corrected and played back to the learner — the same corrective principle used presently by instructors but performed manually. Given that there are millions of language learners, advancements in this area could potentially have global impact.

- If we are successful, accents will be able to be detected in a short amount of time and the speech engine will be able to be switched from standard to accent-adapted. For the case of customer calls, more information may be recognized and collected by the improved speech recognition engine and the call from the accented speaker may be directed to an agent with the similar language origin for better service. This technology could have widespread use by companies operating in global markets.

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Warschauer, "Computer assisted language learning: An introduction (web version, from www. ict4lt. org)," *Multimedia language teaching*, no. Taylor 1980, pp. 3–20, 1996.

[2] H. Franco, L. Neumeyer, M. Ramos, and H. Bratt, "Automatic detection of phone-level mispronunciation for language learning," in *Proc. Eurospeech*, vol. 99, pp. 851–854, Citeseer, 1999.

[3] T. Cincarek, R. Gruhn, C. Hacker, E. Noth, and S. Nakamura, "Automatic pronunciation scoring of words and sentences independent from the non-native's first language," *Computer Speech & Language*, vol. 23, no. 1, pp. 65–88, 2009.

[4] T. Schultz and A. Waibel, "Language-independent and language-adaptive acoustic modeling for speech recognition," *Speech Communication*, vol. 35, no. 1, pp. 31–52, 2001.

[5] X. Huang and K. Lee, "On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 1, no. 2, pp. 150–157, 1993.

[6] Z. Ge, S. R. Sharma, and M. J. Smith, "Adaptive frequency cepstral coefficients for word mispronunciation detection," in *Image and Signal Processing (CISP), 2011 4th International Congress on*, vol. 5, pp. 2388–2391, IEEE, 2011.

[7] S. R. S. Zhenhao Ge and M. J. Smith, "Improving mispronunciation detection using adaptive frequency scale," *Computers & Electrical Engineering*, vol. 39, no. 5, pp. 1464 – 1472, 2013.

[8] Z. Ge, S. R. Sharma, and M. J. Smith, "Pca method for automated detection of mispronounced words," in *SPIE Defense, Security, and Sensing*, pp. 80581D–80581D, International Society for Optics and Photonics, 2011.

[9] Z. Ge, S. R. Sharma, and M. J. Smith, "Pca/lda approach for text-independent speaker recognition," in *SPIE Defense, Security, and Sensing*, pp. 840108–840108, International Society for Optics and Photonics, 2012.

[10] A. Ito, Y. Lim, M. Suzuki, and S. Makino, "Pronunciation error detection for computer-assisted language learning system based on error rule clustering using a decision tree," *Acoustical science and technology*, vol. 28, no. 2, pp. 131–133, 2007.

[11] S. Witt and S. Young, "Phone-level pronunciation scoring and assessment for interactive language learning," *Speech communication*, vol. 30, no. 2-3, pp. 95–108, 2000.

[12] H. Jiang, "Confidence measures for speech recognition: A survey," *Speech communication*, vol. 45, no. 4, pp. 455–470, 2005.

[13] A. Ganapathiraju, J. Hamaker, and J. Picone, "Applications of support vector machines to speech recognition," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2348–2355, 2004.

[14] S. Wei, G. Hu, Y. Hu, and R. Wang, "A new method for mispronunciation detection using support vector machine based on pronunciation space models," *Speech Communication*, vol. 51, no. 10, pp. 896–905, 2009.

[15] C. Hacker, T. Cincarek, A. Maier, A. Hebler, and E. Noth, "Boosting of prosodic and pronunciation features to detect mispronunciations of non-native children," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–197, IEEE.

[16] H. Franco, H. Bratt, R. Rossier, V. Rao Gadde, E. Shriberg, V. Abrash, and K. Precoda, "Eduspeak®: A speech recognition and pronunciation scoring toolkit for computer-aided language learning applications," *Language Testing*, vol. 27, no. 3, p. 401, 2010.

[17] K. Bartkova and D. Jouvet, "Automatic detection of foreign accent for automatic speech recognition," *ICPhS XVI. ID1126. Saarbrucken*, pp. 2185–2188, 2007.

[18] M. Raab, R. Gruhn, and E. Noeth, "Non-native speech databases," in *Automatic Speech Recognition & Understanding, 2007. ASRU. IEEE Workshop on*, pp. 413–418, IEEE, 2007.

[19] J. Hieronymus, "Ascii phonetic symbols for the worldâĂŹs languages: Worldbet," *Journal of the International Phonetic Association*, vol. 23, 1993.

[20] L. Rabiner and B. Juang, "A tutorial on hidden markov models," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[21] M. Xu, L. Duan, J. Cai, L. Chia, C. Xu, and Q. Tian, "Hmm-based audio keyword generation," *Advances in Multimedia Information Processing-PCM 2004*, pp. 566–574, 2005.

[22] F. Fritsch and R. Carlson, "Monotone piecewise cubic interpolation," *Society for Industrial and Applied Mathematics*, vol. 2, 1980.

[23] S. Young, et al., *The HTK Book*. 3.4 ed., 2009.

[24] M. Slaney, "Auditory toolbox." Technical Report No.1998-010, Interval Research Corporation.

[25] E. Zwicker, "Subdivision of the audible frequency range into critical bands," *The Journal of the Acoustical Society of America*, vol. 33, 1961.

[26] L. Sirovich and M. Kirby, "A low-dimensional procedure for the characterization of human faces," *The Journal of the Optical Society of America*, vol. 4, pp. 519–524, 1987.

[27] M. Kirby and L. Sirovich, "Application of the karhunen-loeve procedure for the characterization of human faces," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, pp. 103–107, 1990.

[28] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 771–720, 1997.

[29] T. Mandal and Q.M.J.Wu, "Face recognition using curvelet based pca," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, 2008.

[30] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[31] J. Shlens, "A tutorial on principal component analysis," tech. rep., Institute for Nonlinear Science, UCSD, 2005.

[32] S. A. Batch, "Principal variance component analysis," August 2010. `http://www.niehs.nih.gov/research/resources/software/pvca`.

[33] R. M. Hammond, *The Sounds of Spanish: Analysis and Application with special reference to American English*. Cascadilla, 2001.

[34] Y. Guo and H. Gao, "A chinese person name recognition system based on agent-based hmm position tagging model," in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, vol. 1, pp. 4069–4072, IEEE.

[35] R. C. Rose and D. B. Paul, "A hidden markov model based keyword recognition system," in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pp. 129–132, IEEE, 1990.

[36] J. Tebelskis, *Speech recognition using neural networks*. PhD thesis, Carnegie Mellon University, 1995.

[37] T. L. Kumar, T. K. Kumar, and K. S. Rajan, "Speech recognition using neural networks," in *2009 International Conference on Signal Processing Systems*, pp. 248–252, IEEE, 2009.

[38] A. Ganapathiraju, J. Hamaker, and J. Picone, "Hybrid svm/hmm architectures for speech recognition.," in *INTERSPEECH*, pp. 504–507, Citeseer, 2000.

[39] N. Smith and M. J. Gales, "Using svms and discriminative models for speech recognition," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 1, pp. I–77, IEEE, 2002.

[40] Y. Gao, B. Ramabhadran, J. Chen, H. Erdogan, and M. Picheny, "Innovative approaches for large vocabulary name recognition," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 1, pp. 53–56, IEEE, 2001.

[41] C. J. Leggetter and P. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models," *Computer Speech & Language*, vol. 9, no. 2, pp. 171–185, 1995.

[42] J.-L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains," *Speech and audio processing, ieee transactions on*, vol. 2, no. 2, pp. 291–298, 1994.

[43] G. Bouselmi, D. Fohr, I. Illina, and J.-P. Haton, "Fully automated non-native speech recognition using confusion-based acoustic model integration and graphemic constraints," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, pp. I–I, IEEE, 2006.

[44] N. Cremelie and L. t. Bosch, "Improving the recognition of foreign names and non-native speech by combining multiple grapheme-to-phoneme converters," in *ISCA Tutorial and Research Workshop (ITRW) on Adaptation Methods for Speech Recognition*, 2001.

[45] X. Li, A. Gunawardana, and A. Acero, "Adapting grapheme-to-phoneme conversion for name recognition," in *Automatic Speech Recognition & Understanding, 2007. ASRU. IEEE Workshop on*, pp. 130–135, IEEE, 2007.

[46] F. Beaufays, A. Sankar, S. Williams, and M. Weintraub, "Learning name pronunciations in automatic speech recognition systems," in *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pp. 233–240, IEEE, 2003.

[47] Wikipedia, "Arpabet," August 2011. `http://en.wikipedia.org/wiki/Arpabet`.

[48] R. Doe, "CMU pronouncing dictionary @ONLINE," 2013.

[49] P. Woodland, J. Odell, V. Valtchev, and S. Young, "Large vocabulary continuous speech recognition using htk," in *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, vol. 2, pp. II–125, Ieee, 1994.

[50] K. Lee, H. Hon, and R. Reddy, "An overview of the sphinx speech recognition system," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 38, no. 1, pp. 35–45, 1990.

[51] A. Lee, T. Kawahara, and K. Shikano, "Julius—an open source real-time large vocabulary recognition engine," in *Seventh European Conference on Speech Communication and Technology*, 2001.

[52] P. Clarkson and R. Rosenfeld, "Statistical language modeling using the cmu-cambridge toolkit," in *Fifth European Conference on Speech Communication and Technology*, 1997.

[53] A. Stolcke, "Srilm-an extensible language modeling toolkit," in *Seventh International Conference on Spoken Language Processing*, 2002.

[54] N. Minematsu, "Yet another acoustic representation of speech sounds," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, vol. 1, pp. I–585, IEEE, 2004.

[55] M. Suzuki, L. Dean, N. Minematsu, and K. Hirose, "Improved structure-based automatic estimation of pronunciation proficiency," *Proc. SLaTE*, vol. 5, 2009.

[56] A. Bhattacharyya, "On a measure of divergence between two multinomial populations," *Sankhyā: The Indian Journal of Statistics (1933-1960)*, vol. 7, no. 4, pp. 401–406, 1946.

[57] T. Giannakopoulos, "A method for silence removal and segmentation of speech signals, implemented in matlab," *University of Athens, Athens*, 2009.

[58] G. Choueiter, G. Zweig, and P. Nguyen, "An empirical study of automatic accent classification," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pp. 4265–4268, IEEE, 2008.

[59] D. Reynolds and R. Rose, "Robust text-independent speaker identification using gaussian mixture speaker models," *Speech and Audio Processing, IEEE Transactions on*, vol. 3, no. 1, pp. 72–83, 1995.

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

[61] N. Kumar and A. G. Andreou, *Investigation of silicon auditory models and generalization of linear discriminant analysis for improved speech recognition*. PhD thesis, Johns Hopkins University, 1997.

[62] Wikipedia, "Linear discriminant analysis," September 2013. `http://en.wikipedia.org/wiki/Linear_discriminant_analysis`.

[63] H. Larochelle and Y. Bengio, "Classification using discriminative restricted boltzmann machines," in *Proceedings of the 25th international conference on Machine learning*, pp. 536–543, ACM, 2008.

APPENDICES

A. PHONEME CONFUSION MATRIX

Figure A.1 demonstrates the acoustic confusion matrix. It is symmetric and smaller values indicate more similar the pairs of phones are. The value of ith row and jth column is the average of the scaled "distances" of all samples of ith phone to the acoustic model of jth phone based on Maximum Likelihood Estimation (MLE), and the matrix is originally non-symmetric. Here it is averaged by its transpose to make it symmetric for computation simplicity.

Figure A.2 demonstrates the linguistic confusion matrix which is also symmetric. It indicates the phoneme clustering in Table 4.4. Value 0 indicate the pair of phones belongs to the same cluster and linguistically confusable, and vise versa.

Figure A.3 demonstrates the union of both acoustic and linguistic confusion matrices computed by Equation (4.3), which gives priority to linguistic confusion values. Once the phones belong to the same linguistic cluster, the overall confusion values will be 0 (most confusable), and acoustic confusion values are only considered when the linguistic confusion values is 1.

Given the unioned confusion matrix in A.3, Table A.1 shows the first cluster of the most similar phones of each target phone as its alternative candidates, by applying k-mean clustering on the whole phone list sorted by similarity.

Table A.1: List of candidate phonemes for each phone

Phone	**Thres-hold**	**No. of Candidates**	**Candidates (Confusion Value)**
aa	6	7	aa(0), ae(0), ah(0), ao(0), aw(0), ay(4.5), ow(5.36)
ae	5	7	aa(0), ae(0), ah(0), ao(0), aw(0), eh(1.605), ih(4.915)
ah	4	6	aa(0), ae(0), ah(0), ao(0), aw(0), ih(3.585)
Continued on next page			

Table A.1 – continued from previous page

Phone	Threshold	No. of Candidates	Candidates (Confusion Value)
ao	4	6	aa(0), ae(0), ah(0), ao(0), aw(0), ow(3.995)
aw	5	5	aa(0), ae(0), ah(0), ao(0), aw(0)
ay	5	5	ay(0), ih(0), iy(0), y(0), aa(4.5)
b	2.5	5	b(0), p(0), v(1.47), d(2.305), dh(2.43)
ch	3	3	ch(0), jh(0), zh(2.875)
d	2	4	d(0), t(0), g(1.785), dh(1.98)
dh	2	3	dh(0), th(0), d(1.98)
eh	5	5	eh(0), ey(0), ae(1.605), ih(2.755), ah(4.28)
er	5	4	er(0), l(0), r(0), ah(4.635)
ey	4	4	eh(0), ey(0), iy(2.21), ih(3.115)
f	1	3	f(0), v(0), s(0.92)
g	2	3	g(0), k(0), d(1.785)
hh	0	1	hh(0)
ih	2	4	ay(0), ih(0), iy(0), y(0)
iy	3	5	ay(0), ih(0), iy(0), y(0), ey(2.11)
jh	3	3	ch(0), jh(0), zh(2.02)
k	2	4	g(0), k(0), p(1.5), t(1.97)
l	0	3	er(0), l(0), r(0)
m	1	1	m(0)
n	1	2	n(0), ng(0)
ng	1	2	n(0), ng(0)
ow	4	4	ow(0), oy(0), l(3.975), ao(3.995)
oy	6	2	ow(0), oy(0)
p	1	2	b(0), p(0)
r	4	3	er(0), l(0), r(0)
s	2	6	s(0), sh(0), z(0), zh(0), f(0.92), th(1.295)
sh	2	4	s(0), sh(0), z(0), zh(0)

Continued on next page

Table A.1 – continued from previous page

Phone	Threshold	No. of Candidates	Candidates (Confusion Value)
t	2	3	d(0), t(0), k(1.97)
th	2	6	dh(0), th(0), s(1.295), f(1.49), p(1.49), z(1.995)
uh	5	3	uh(0), uw(0), w(0)
uw	1	3	uh(0), uw(0), w(0)
v	1	2	f(0), v(0)
w	4	4	uh(0), uw(0), w(0), l(3.305)
y	5	4	ay(0), ih(0), iy(0), y(0)
z	2	5	s(0), sh(0), z(0), zh(0), th(1.995)
zh	3	6	s(0), sh(0), z(0), zh(0), jh(2.02), ch(2.875)

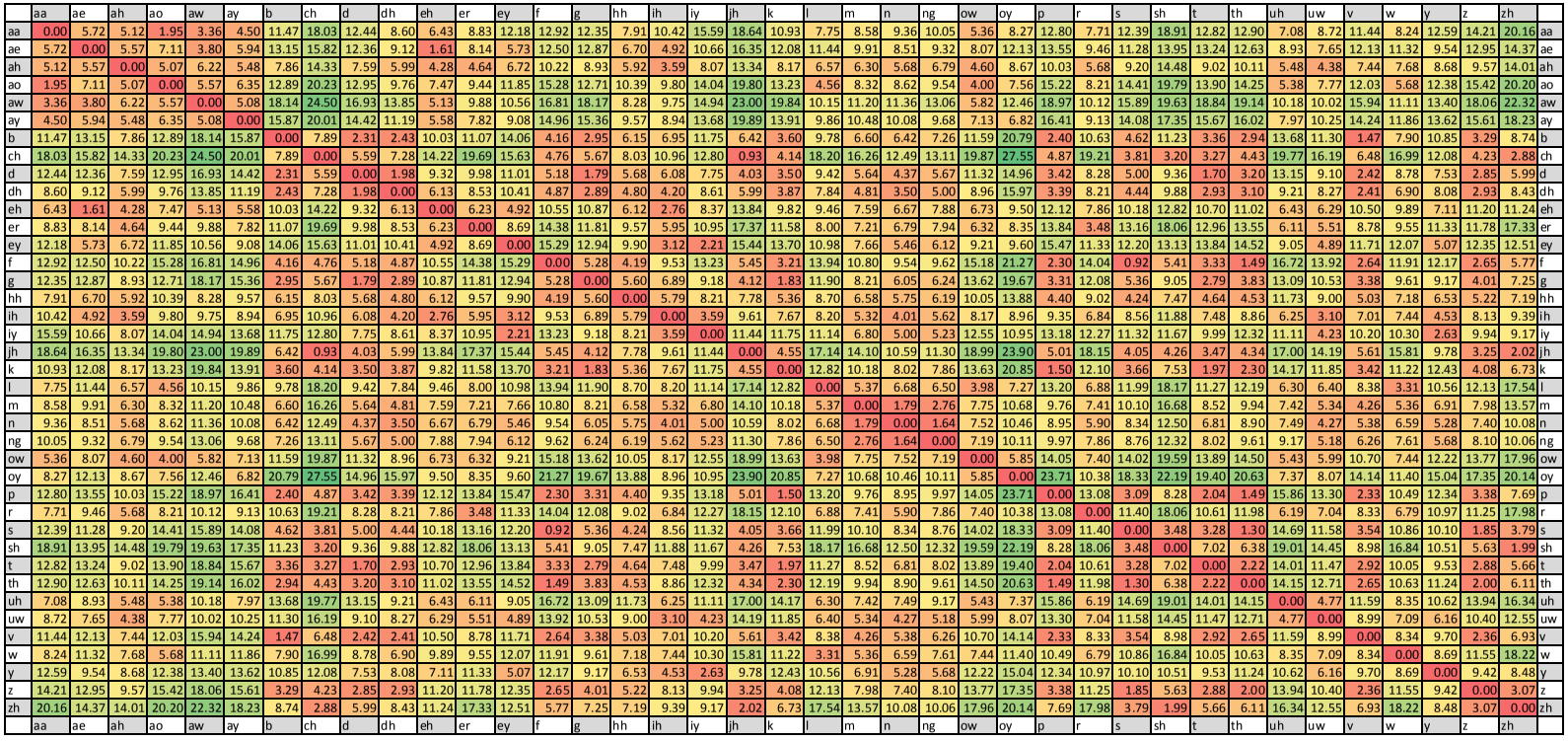


Fig. A.1. Acoustic confusion matrix with color scales

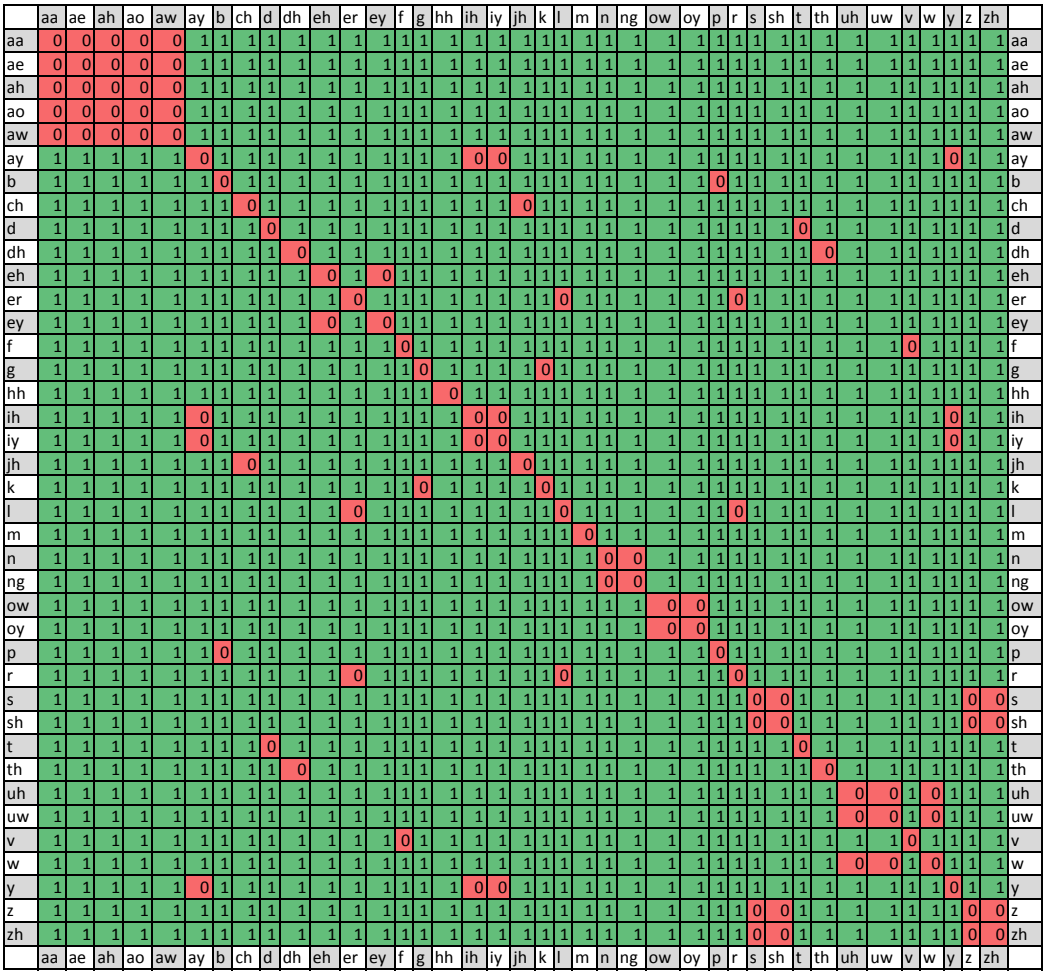
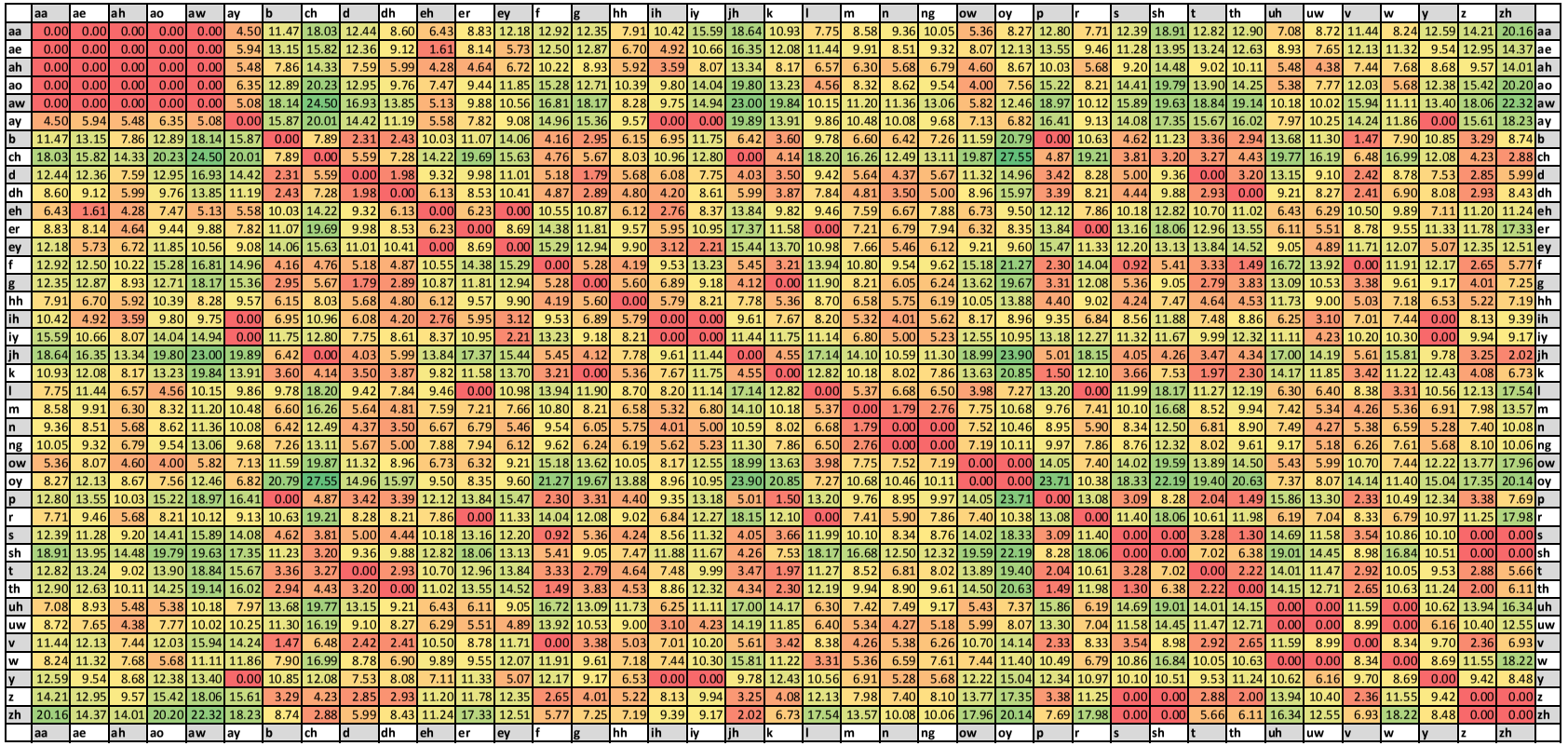


Fig. A.2. Linguistic confusion matrix with color Scales



	aa	ae	ah	ao	aw	ay	b	ch	d	dh	eh	er	ey	f	g	hh	ih	iy	jh	k	l	m	n	ng	ow	oy	p	r	s	sh	t	th	uh	uw	v	w	y	z	zh	
aa	0.00	0.00	0.00	0.00	0.00	4.50	11.47	18.03	12.44	8.60	6.43	8.83	12.18	12.92	12.35	7.91	10.42	15.59	18.64	10.93	7.75	8.58	9.36	10.05	5.36	8.27	12.80	7.71	12.39	18.91	12.82	12.90	7.08	8.72	11.44	8.24	12.59	14.21	20.16	aa
ae	0.00	0.00	0.00	0.00	0.00	5.94	13.15	15.82	12.36	9.12	1.61	8.14	5.73	12.50	12.87	6.70	4.92	10.66	16.35	12.08	11.44	9.91	8.51	9.32	8.07	12.13	13.55	9.46	11.28	13.95	13.24	12.63	8.93	7.65	12.13	11.32	9.54	12.95	14.37	ae
ah	0.00	0.00	0.00	0.00	0.00	5.48	7.86	14.33	7.59	5.99	4.28	4.64	6.72	10.22	8.93	5.92	3.59	8.07	13.34	8.17	6.57	6.30	5.68	6.79	4.60	8.67	10.03	5.68	9.20	14.48	9.02	10.11	5.48	4.38	7.44	7.68	8.68	9.57	14.01	ah
ao	0.00	0.00	0.00	0.00	0.00	6.35	12.89	20.23	12.95	9.76	7.47	9.44	11.85	15.28	12.71	10.39	9.80	14.04	19.80	13.23	4.56	8.32	8.62	9.54	4.00	7.56	15.22	8.21	14.41	19.79	13.90	14.25	5.38	7.77	12.03	5.68	12.38	15.42	20.20	ao
aw	0.00	0.00	0.00	0.00	0.00	5.08	18.14	24.50	16.93	13.85	5.13	9.88	10.56	16.81	18.17	8.28	9.75	14.94	23.00	19.84	10.15	11.20	11.36	13.06	5.82	12.46	18.97	10.12	15.89	19.63	18.84	19.14	10.18	10.02	15.94	11.11	13.40	18.06	22.32	aw
ay	4.50	5.94	5.48	6.35	5.08	0.00	15.87	20.01	14.42	11.19	5.58	7.82	9.08	14.96	15.36	9.57	0.00	0.00	19.89	13.91	9.86	10.48	10.08	9.68	7.13	6.82	16.41	9.13	14.08	17.35	15.67	16.02	7.97	10.25	14.24	11.86	0.00	15.61	18.23	ay
b	11.47	13.15	7.86	12.89	18.14	15.87	0.00	7.89	2.31	2.43	10.03	11.07	14.06	4.16	2.95	6.15	6.95	11.75	6.42	3.60	9.78	6.60	6.42	7.26	11.59	20.79	0.00	10.63	4.62	11.23	3.36	2.94	13.68	11.30	1.47	7.90	10.85	3.29	8.74	b
ch	18.03	15.82	14.33	20.23	24.50	20.01	7.89	0.00	5.59	7.28	14.22	19.69	15.63	4.76	5.67	8.03	10.96	12.80	0.00	4.14	18.20	16.26	12.49	13.11	19.87	27.55	4.87	19.21	3.81	3.20	3.27	4.43	19.77	16.19	6.48	16.99	12.08	4.23	2.88	ch
d	12.44	12.36	7.59	12.95	16.93	14.42	2.31	5.59	0.00	1.98	9.32	9.98	11.01	5.18	1.79	5.68	6.08	7.75	4.03	3.50	9.42	5.64	4.37	5.67	11.32	14.96	3.42	8.28	5.00	9.36	0.00	3.20	13.15	9.10	2.42	8.78	7.53	2.85	5.99	d
dh	8.60	9.12	5.99	9.76	13.85	11.19	2.43	7.28	1.98	0.00	6.13	8.53	10.41	4.87	2.89	4.80	4.20	8.61	5.99	3.87	7.84	4.81	3.50	5.00	8.96	15.97	3.39	8.21	4.44	9.88	2.93	0.00	9.21	8.27	2.41	6.90	8.08	2.93	8.43	dh
eh	6.43	1.61	4.28	7.47	5.13	5.58	10.03	14.22	9.32	6.13	0.00	6.23	0.00	10.55	10.87	6.12	2.76	8.37	13.84	9.82	9.46	7.59	6.67	7.88	6.73	9.50	12.12	7.86	10.18	12.82	10.70	11.02	6.43	6.29	10.50	9.89	7.11	11.20	11.24	eh
er	8.83	8.14	4.64	9.44	9.88	7.82	11.07	19.69	9.98	8.53	6.23	0.00	8.69	14.38	11.81	9.57	5.95	10.95	17.37	11.58	0.00	7.21	6.79	7.94	6.32	8.35	13.84	0.00	13.16	18.06	12.96	13.55	6.11	5.51	8.78	9.55	11.33	11.78	17.33	er
ey	12.18	5.73	6.72	11.85	10.56	9.08	14.06	15.63	11.01	10.41	0.00	8.69	0.00	15.29	12.94	9.90	3.12	2.21	15.44	13.70	10.98	7.66	5.46	6.12	9.21	9.60	15.47	11.33	12.20	13.13	13.84	14.52	9.05	4.89	11.71	12.07	5.07	12.35	12.51	ey
f	12.92	12.50	10.22	15.28	16.81	14.96	4.16	4.76	5.18	4.87	10.55	14.38	15.29	0.00	5.28	4.19	9.53	13.23	5.45	3.21	13.94	10.80	9.54	9.62	15.18	21.27	2.30	14.04	0.92	5.41	3.33	1.49	16.72	13.92	0.00	11.91	12.17	2.65	5.77	f
g	12.35	12.87	8.93	12.71	18.17	15.36	2.95	5.67	1.79	2.89	10.87	11.81	12.94	5.28	0.00	5.60	6.89	9.18	4.12	0.00	11.90	8.21	6.05	6.24	13.62	19.67	3.31	12.08	5.36	9.05	2.79	3.83	13.09	10.53	3.38	9.61	9.17	4.01	7.25	g
hh	7.91	6.70	5.92	10.39	8.28	9.57	6.15	8.03	5.68	4.80	6.12	9.57	9.90	4.19	5.60	0.00	5.79	8.21	7.78	5.36	8.70	6.58	5.75	6.19	10.05	13.88	4.40	9.02	4.24	7.47	4.64	4.53	11.73	9.00	5.03	7.18	6.53	5.22	7.19	hh
ih	10.42	4.92	3.59	9.80	9.75	0.00	6.95	10.96	6.08	4.20	2.76	5.95	3.12	9.53	6.89	5.79	0.00	0.00	9.61	7.67	8.20	5.32	4.01	5.62	8.17	8.96	9.35	6.84	8.56	11.88	7.48	8.86	6.25	3.10	7.01	7.44	0.00	8.13	9.39	ih
iy	15.59	10.66	8.07	14.04	14.94	0.00	11.75	12.80	7.75	8.61	8.37	10.95	2.21	13.23	9.18	8.21	0.00	0.00	11.44	11.75	11.14	6.80	5.00	5.23	12.55	10.95	13.18	12.27	11.32	11.67	9.99	12.32	11.11	4.23	10.20	10.30	0.00	9.94	9.17	iy
jh	18.64	16.35	13.34	19.80	23.00	19.89	6.42	0.00	4.03	5.99	13.84	17.37	15.44	5.45	4.12	7.78	9.61	11.44	0.00	4.55	17.14	14.10	10.59	11.30	18.99	23.90	5.01	18.15	4.05	4.26	3.47	4.34	17.00	14.19	5.61	15.81	9.78	3.25	2.02	jh
k	10.93	12.08	8.17	13.23	19.84	13.91	3.60	4.14	3.50	3.87	9.82	11.58	13.70	3.21	0.00	5.36	7.67	11.75	4.55	0.00	12.82	10.18	8.02	7.86	13.63	20.85	1.50	12.10	3.66	7.53	1.97	2.30	14.17	11.85	3.42	11.22	12.43	4.08	6.73	k
l	7.75	11.44	6.57	4.56	10.15	9.86	9.78	18.20	9.42	7.84	9.46	0.00	10.98	13.94	11.90	8.70	8.20	11.14	17.14	12.82	0.00	5.37	6.68	6.50	3.98	7.27	13.20	0.00	11.99	18.17	11.27	12.19	6.30	6.40	8.38	3.31	10.56	12.13	17.54	l
m	8.58	9.91	6.30	8.32	11.20	10.48	6.60	16.26	5.64	4.81	7.59	7.21	7.66	10.80	8.21	6.58	5.32	6.80	14.10	10.18	5.37	0.00	1.79	2.76	7.75	10.68	9.76	7.41	10.10	16.68	8.52	9.94	7.42	5.34	4.26	5.36	6.91	7.98	13.57	m
n	9.36	8.51	5.68	8.62	11.36	10.08	6.42	12.49	4.37	3.50	6.67	6.79	5.46	9.54	6.05	5.75	4.01	5.00	10.59	8.02	6.68	1.79	0.00	0.00	7.52	10.46	8.95	5.90	8.34	12.50	6.81	8.90	7.49	4.27	5.38	6.59	5.28	7.40	10.08	n
ng	10.05	9.32	6.79	9.54	13.06	9.68	7.26	13.11	5.67	5.00	7.88	7.94	6.12	9.62	6.24	6.19	5.62	5.23	11.30	7.86	6.50	2.76	0.00	0.00	7.19	10.11	9.97	7.86	8.76	12.32	8.02	9.61	9.17	5.18	6.26	7.61	5.68	8.10	10.06	ng
ow	5.36	8.07	4.60	4.00	5.82	7.13	11.59	19.87	11.32	8.96	6.73	6.32	9.21	15.18	13.62	10.05	8.17	12.55	18.99	13.63	3.98	7.75	7.52	7.19	0.00	0.00	14.05	7.40	14.02	19.59	13.89	14.50	5.43	5.99	10.70	7.44	12.22	13.77	17.96	ow
oy	8.27	12.13	8.67	7.56	12.46	6.82	20.79	27.55	14.96	15.97	9.50	8.35	9.60	21.27	19.67	13.88	8.96	10.95	23.90	20.85	7.27	10.68	10.46	10.11	0.00	0.00	23.71	10.38	18.33	22.19	19.40	20.63	7.37	8.07	14.14	11.40	15.04	17.35	20.14	oy
p	12.80	13.55	10.03	15.22	18.97	16.41	0.00	4.87	3.42	3.39	12.12	13.84	15.47	2.30	3.31	4.40	9.35	13.18	5.01	1.50	13.20	9.76	8.95	9.97	14.05	23.71	0.00	13.08	3.09	8.28	2.04	1.49	15.86	13.30	2.33	10.49	12.34	3.38	7.69	p
r	7.71	9.46	5.68	8.21	10.12	9.13	10.63	19.21	8.28	8.21	7.86	0.00	11.33	14.04	12.08	9.02	6.84	12.27	18.15	12.10	0.00	7.41	5.90	7.86	7.40	10.38	13.08	0.00	11.40	18.06	10.61	11.98	6.19	7.04	8.33	6.79	10.97	11.25	17.98	r
s	12.39	11.28	9.20	14.41	15.89	14.08	4.62	3.81	5.00	4.44	10.18	13.16	12.20	0.92	5.36	4.24	8.56	11.32	4.05	3.66	11.99	10.10	8.34	8.76	14.02	18.33	3.09	11.40	0.00	0.00	3.28	1.30	14.69	11.58	3.54	10.86	10.10	0.00	0.00	s
sh	18.91	13.95	14.48	19.79	19.63	17.35	11.23	3.20	9.36	9.88	12.82	18.06	13.13	5.41	9.05	7.47	11.88	11.67	4.26	7.53	18.17	16.68	12.50	12.32	19.59	22.19	8.28	18.06	0.00	0.00	7.02	6.38	19.01	14.45	8.98	16.84	10.51	0.00	0.00	sh
t	12.82	13.24	9.02	13.90	18.84	15.67	3.36	3.27	0.00	2.93	10.70	12.96	13.84	3.33	2.79	4.64	7.48	9.99	3.47	1.97	11.27	8.52	6.81	8.02	13.89	19.40	2.04	10.61	3.28	7.02	0.00	2.22	14.01	11.47	2.92	10.05	9.53	2.88	5.66	t
th	12.90	12.63	10.11	14.25	19.14	16.02	2.94	4.43	3.20	0.00	11.02	13.55	14.52	1.49	3.83	4.53	8.86	12.32	4.34	2.30	12.19	9.94	8.90	9.61	14.50	20.63	1.49	11.98	1.30	6.38	2.22	0.00	14.15	12.71	2.65	10.63	11.24	2.00	6.11	th
uh	7.08	8.93	5.48	5.38	10.18	7.97	13.68	19.77	13.15	9.21	6.43	6.11	9.05	16.72	13.09	11.73	6.25	11.11	17.00	14.17	6.30	7.42	7.49	9.17	5.43	7.37	15.86	6.19	14.69	19.01	14.01	14.15	0.00	0.00	11.59	0.00	10.62	13.94	16.34	uh
uw	8.72	7.65	4.38	7.77	10.02	10.25	11.30	16.19	9.10	8.27	6.29	5.51	4.89	13.92	10.53	9.00	3.10	4.23	14.19	11.85	6.40	5.34	4.27	5.18	5.99	8.07	13.30	7.04	11.58	14.45	11.47	12.71	0.00	0.00	8.99	0.00	6.16	10.40	12.55	uw
v	11.44	12.13	7.44	12.03	15.94	14.24	1.47	6.48	2.42	2.41	10.50	8.78	11.71	0.00	3.38	5.03	7.01	10.20	5.61	3.42	8.38	4.26	5.38	6.26	10.70	14.14	2.33	8.33	3.54	8.98	2.92	2.65	11.59	8.99	0.00	8.34	9.70	2.36	6.93	v
w	8.24	11.32	7.68	5.68	11.11	11.86	7.90	16.99	8.78	6.90	9.89	9.55	12.07	11.91	9.61	7.18	7.44	10.30	15.81	11.22	3.31	5.36	6.59	7.61	7.44	11.40	10.49	6.79	10.86	16.84	10.05	10.63	0.00	0.00	8.34	0.00	8.69	11.55	18.22	w
y	12.59	9.54	8.68	12.38	13.40	0.00	10.85	12.08	7.53	8.08	7.11	11.33	5.07	12.17	9.17	6.53	0.00	0.00	9.78	12.43	10.56	6.91	5.28	5.68	12.22	15.04	12.34	10.97	10.10	10.51	9.53	11.24	10.62	6.16	9.70	8.69	0.00	9.42	8.48	y
z	14.21	12.95	9.57	15.42	18.06	15.61	3.29	4.23	2.85	2.93	11.20	11.78	12.35	2.65	4.01	5.22	8.13	9.94	3.25	4.08	12.13	7.98	7.40	8.10	13.77	17.35	3.38	11.25	0.00	0.00	2.88	2.00	13.94	10.40	2.36	11.55	9.42	0.00	0.00	z
zh	20.16	14.37	14.01	20.20	22.32	18.23	8.74	2.88	5.99	8.43	11.24	17.33	12.51	5.77	7.25	7.19	9.39	9.17	2.02	6.73	17.54	13.57	10.08	10.06	17.96	20.14	7.69	17.98	0.00	0.00	5.66	6.11	16.34	12.55	6.93	18.22	8.48	0.00	0.00	zh
	aa	ae	ah	ao	aw	ay	b	ch	d	dh	eh	er	ey	f	g	hh	ih	iy	jh	k	l	m	n	ng	ow	oy	p	r	s	sh	t	th	uh	uw	v	w	y	z	zh	

Fig. A.3. Union of linguistic and acoustic confusion matrices with colour Scales

B. HLDA IMPLEMENTATION IN MATLAB

The implementation of HLDA in developing discriminatively optimized acoustic features for accent classification is developed in MATLAB. The scripts are originally from Dr. Kumar's HLDA thesis [61], and are now improved in the following three aspects:

1. simplify the original scripts by:
   - combine similar scripts with different assumptions of data class covariance matrices;
   - reorganize routines in scripts by removing redundant variables and combine similarly repeated routines;
   - take advantage of new MATLAB version with cell structure and limit the number of global variables.
2. generalize the code for classes with unequal number of data samples;
3. enable HLDA option such as frame size and deal with memory efficiency in computing data global mean and between-class scatter.

B.1 Outline of the MATLAB scripts

1. `main_setup.m` and `main_solve.m` set up experiments to demonstrate the performance of HLDA with two different assumptions of class covariances (type 1 and type 2), compared with LDA (type 1);
2. `main_hlda.m` is an example to transform original data to HLDA data using HLDA;
3. `main_setup.m` and `main_solve.m` are the simplified and re-organized version from Dr. Kumar's original MATLAB scripts, and `main_hlda.m` is my implementation using several scripts from Dr. Kumar.

B.2 Detail Description of the MATLAB Scripts

1. `main_setup.m`: generate random Gaussian training and testing data and compute the sufficient statistic for Heteroscedastic LDA (HLDA)
   a. `makedata.m`: generate high-dimensional training (x) and testing (T_x) data (dimension $p = 5$) with uni-model Gaussian distribution with desired properties through classes:
      i. type 1: equal covariance (could be full rank);
      ii. type 2: unequal covariance, but diagonal;
      iii. type 3: unequal covariance and full rank.
   b. `initialize.m`: compute sufficient statistic of the randomly generated training data x for HLDA [refer to Section 2.1 on Dr. Kumar's thesis]:
      i. X_b: global mean;
      ii. X_{jb}: class mean;
      iii. TB: between-class scatter;
      iv. W_jB: within-class scatters for each class;
      v. WB: within-class scatter for entire training data.
2. `main_solve.m`: using training and testing data generated in `main_setup.m`, compute the HLDA transform matrix theta in each type and obtain the classification performance for the case $p = 2$ and $p = 5$ (no dimension reduction)[refer to Chapter 5 on Dr. Kumar's thesis]
   a. `problem.m`: find the HLDA transform matrix theta using Steepest Descent (SD) optimization, based on the type of assumption in covariance matrices of data in classes and various optimization options;
   b. `fminsd.m`(main Steepest Descent (SD) optimization script): use SD optimization to find the minimum of the supplied function;
   c. `loglik.m` and `gradient.m`: compute log likelihood $L_E(\theta|x)$ without the constant terms and compute the gradient of $-L_E(\theta|x)$.
3. `main_hlda.m`: main function to load feature and transform original feature to HLDA feature using SD optimization with various assumptions in class covariance and option settings in optimization
   a. `hlda.m`: find the HLDA transform matrix using the original feature (a combination of `initialize.m` and `fminsd.m` with `loglik.m` and `gradient.m`).

B.3 Structure of the MATLAB Scripts

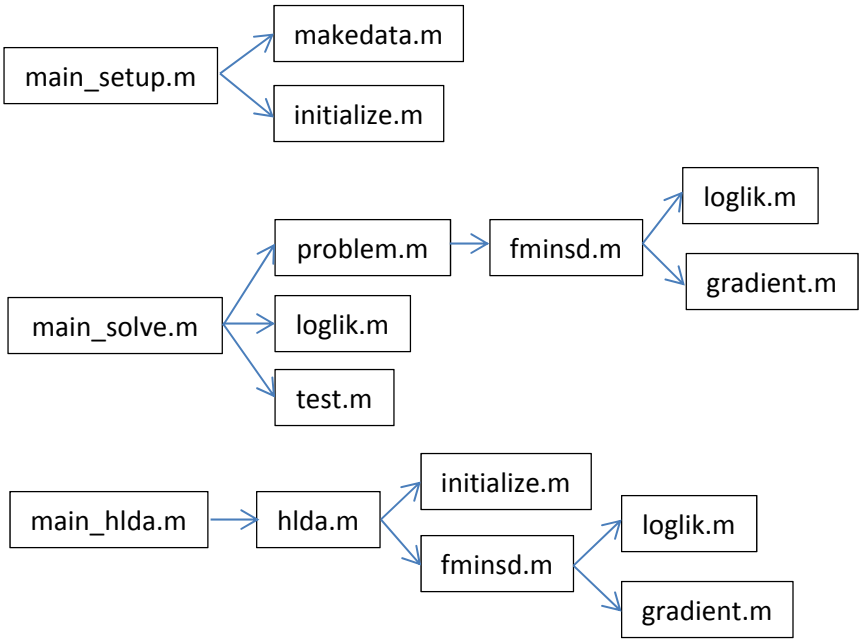


Fig. B.1. Structure of the MATALB scripts

B.4 Correspondences between the New HLDA Scripts and the Original Scripts

1. `main_setup.m` ⇒ `setup.m`;
2. `main_solve.m` ⇒ `solve.m`;
3. `problem.m` ⇒ `problem1.m, problem2.m, problem3.m`;
4. `loglik.m` ⇒ `m1.m, m2.m, m3.m`;
5. `gradient.m` ⇒ `g1.m, g2.m, g3.m`;
6. `test.m` ⇒ `test1.m, test2.m, test3.m`;
7. The remaining scripts share the same names as the original scripts.

B.5 Examples of Data Generated and Used in Experiments

1. `data_equal1.mat`(`data_equal2.mat`)/`setup_equal1.mat`(`setup_equal2.mat`): data with equally 200 samples in 4 classes;
2. `data_unequal1.mat`(`data_unequal2.mat`)/`setup_unequal2.mat` (`setup_unequal2.mat`): data with unequal number of samples in 4 classes: 200, 300, 100 and 200;
3. `data_unequal3.mat`/`setup_unequal3.mat`: data with unequal number of samples in 4 classes: 200, 201, 199 and 200.

B.6 List of the MATLAB Scripts

The improved MATLAB scripts for HLDA listed below are included for reference:

1. `main_setup.m`;
2. `makedata.m`;
3. `initialize2.m`;
4. `main_solve.m`;
5. `problem.m`;
6. `fminsd.m`;
7. `loglik.m`;
8. `gradient.m`;
9. `test.m`;
10. `main_hlda.m`;
11. `hlda.m`.

codes/hlda2/main_setup.m

```
% This program geneates random data and initializes all the variables for
% later use. You can use this program to generate a random problem.
%
% Children: makedata, initialize
%
% Variables:
% - x{J}[NJ(J),n]: training data
% - Tx{J}[NJ(J)*50,n]: testing data
% - NJ: # of training data in each class
% - TNJ: # of testing data in each class
% - WB: global mean
% - Xjb: class means
% - TB: between-class scatter
% - WjB: within-class scatters for each class
% - WB: within-class scatter for entire training data
%
% Author: Nagendra Kumar
% Modified by Zhenhao Ge, 2012-12-09

clear all
global WjB WB TB Xjb Xb

% feature dimension
n = 5;
```

```
25
% The dimension of the sub-space that contains discrimination information
p0 = 2;

% The random problem will correspond to one o f the three types :
%    1. All class variances are the same ( section 5.1.3)
%    2. Class variances are diagonalizable by the same transformation
%       ( section 5.1.2)
%    3. Class variances are un-equal ( section 5.1.1)

type = 3;

gendata = 1; % option to use new generated data or existed data
useequal = 0;
if gendata == 1
    % set the points per class and total points for training and testing
    NJ = [100; 150; 200; 250; 300];
    J = length(NJ); % # of classes
    N = sum(NJ); % total training points
    TNJ= round(max(10000,N)*NJ/mean(NJ));
    [x,Tx] = makedata(n,p0,NJ,TNJ,type);
    save(['data_', datestr(now,30)], 'x', 'Tx');
elseif gendata == 0
    if useequal == 1
        load('data_equal2','x','Tx');
        x = {x(1:200,:);x(201:400,:);x(401:600,:);x(601:800,:)};
        Tx = {Tx(1:10000,:);Tx(10001:20000,:);...
            Tx(20001:30000,:);Tx(30001:40000,:)};
    else
        load('data_unequal3_200-201-199-200','x','Tx');
    end
end

%   Initialization
c = 1; % frame size factor
% [WB,WjB,TB,Xjb,Xb] = initialize(x);
[WB,WjB,TB,Xjb,Xb] = initialize2(x,c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% These are Numerical optimization option %
options = foptions;
% Provide a feedback every options(1) iterations
options(1) = 200;
%options(1) = 0;
% Minimum Step Size
options(2) = 1e-8;
% Minimum improvement at every iteration
options(3) = 1e-8;
options(6) = 1;
options(9) = 0;
% options(9) = 1;
options(13) = 0;
% Maximum number of iterations
options(14) = 1000000;
options(16) = 1e-11;
options(17) = 1e-3;
% Initial guess of step-size
options(18) = 0.001;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

save ([ 'setup_', datestr (now ,30) ], 'x', 'Tx', 'c', 'p0', 'type', 'Xb', ...
    'Xjb', 'WjB', 'WB', 'TB', ' options ');
```

codes/hlda2/makedata.m

```
function [x,Tx, theta ,mu] = makedata(n,p,NJ,TNJ,type)
% Program for creating random data, it assumes that you have specified your
% requirements using the "setup" script
%
% Variables :
% - n: feature dimension
% - J: # of classes
%
% Usage: makedata
% Programmar: Nagendra Kumar
% Modified by Zhenhao Ge, 2012-12-09

% % save the current generator settings in s
% s = rng(' shuffle ');

theta = rand(n,n);
Ithetat = inv( theta ');

J = length(NJ);
mu = zeros(n,J);
% x = zeros(ppc*J,n);
% Tx = zeros(Tppc*J,n);
x = cell (J,1);
Tx = cell (J,1);
for j = 1:J
    mu(:,j) = Ithetat (:,1:p)*randn(p,1)*2;
    x{j} = ones(NJ(j),1)*mu(:,j)';
    Tx{j} = ones(TNJ(j),1)*mu(:,j)';
end

% A way to decide the ratio of the variance between the rejected and
% retained sub-spaces
% alpha = rand(1,1);
alpha = 1;
beta = 5*alpha;
Srej = beta*randn(n-p,n-p);

if type == 3 % section 5.1.1 (full rank)
   for j = 1:J
       Sj = alpha*randn(p,p); % everytime different
       % every section of x = x (orginal) + variance around it
       x{j} = x{j} + [randn(NJ(j),p)*Sj, randn(NJ(j),n-p)*Srej]* Ithetat ';
       Tx{j} = Tx{j} + [randn(TNJ(j),p)*Sj, randn(TNJ(j),n-p)*Srej]* Ithetat ';
   end
 elseif type == 2 % section 5.1.2 (diagonal)
   for j = 1:J
       Sj = alpha*randn(p,p);
       Sj = diag(diag(Sj));
       x{j} = x{j} + [randn(NJ(j),p)*Sj, randn(NJ(j),n-p)*Srej]* Ithetat ';
       Tx{j} = Tx{j} + [randn(TNJ(j),p)*Sj, randn(TNJ(j),n-p)*Srej]* Ithetat ';
   end
```

```
elseif type == 1 % section 5.1.3 (equal)
  Sj = randn(p,p);
  for j = 1:J
      x{j} = x{j} + [randn(NJ(j),p)*Sj, randn(NJ(j),n-p)*Srej]*Ithetat';
      Tx{j} = Tx{j} + [randn(TNJ(j),p)*Sj, randn(TNJ(j),n-p)*Srej]*Ithetat';
  end
end
```

codes/hlda2/initialize2.m

```
function [WB,WjB,TB,Xjbc,Xbc] = initialize2(x,c)
% This program computes the sufficient statistic for Heteroscedastic Linear
% Discriminant Analysis.
%
% The initial guess for theta can be Identity matrix.
%
% Imporved from initialize.m with 1) consideration of memory use and frame
% size (results slightly different in the computation of Xb and TB), and 2)
% add frame size factor c to overlap data for potential accuracy
% improvement
%
% Variables:
%   x{J}[NJ(j),n] - data
%   xv[NJ{j},nc] - global mean centralized data for current segment
%   c - # of frames add on each side of original frame (frame size = 2*c+1)
%   n - data dimension
%   nc - data dimension with frame factor
%   N - total # of samples
%   Nc - total # of samples with frame factor
%   J - # of classes
%   NJ[J,1] - # of samples in each class
%   Xjb[n,J] - class mean in columns
%   Xjbc[nc,J] - class mean in columns with frame factor
%   Xb[n,1] - global mean
%   Xbc[nc,1] - global mean with frame factor
%   Wj{J,1}[n,n] - overall within-class scatter for each class
%   Wj2[n,n,J] - alternative Wj with different data format
%   WjB{J,1}[n,n] - normalized within-class scatter for each class
%   WB[n,n] - within-class scatter
%   T[nc,nc] - overall between-class scatter
%   TB[n,n] - between-class scatter
%
% Usage: Initialize
% Programmer: Zhenhao Ge, 2012-12-09
% Note: alternative script of initialize.m from Nagendra Kumar

if nargin < 2, c = 0; end

% # of total points for each classes ('NJ') and # of classes
[NJ,n] = cellfun(@size,x);
J = length(NJ);

% feature dimension ('n')
if range(n) ~= 0, error('feature dimensions inconsistent among classes!');
else n = n(1); end

% class means ('Xjb')
Xjb = cell2mat(cellfun(@mean, x, 'UniformOutput', 0))';
```

```
% global mean ( alternative approach , faster with ignorable computational
% truncation errors )
Xb = Xjb*NJ./sum(NJ);

% extend frame with frame size (' fs ')
fs = 2*c+1;
nc = n*fs;
NJc = NJ-fs+1;
Nc = sum(NJc);
Xjbc = repmat(Xjb, fs ,1);
Xbc = repmat(Xb, fs ,1);

% find between-class & within-class scatter TB & WB for each class (eq. 2.5)
T = zeros(nc);
Wj = cell(J,1);
WjB = cell(J,1);
for j = 1:J

    if fs == 1

        % Kumar's approach to find between-class scatter
        xv = x{j} - ones(NJc(j),1)*Xbc';
        T = T + xv'*xv;

        % find within-class scatter
        xj = x{j} - ones(NJc(j),1)*Xjbc(:, j)';
        Wj{j} = xj'*xj;

    elseif fs > 1

        nSeg = round( linspace (1, NJ(j), fs));
        nSeg(end) = nSeg(end)+1;
        Wj{j} = zeros(nc);

        for i = 1: fs-1

            % find current overlaped data
            xi = matola(x{j}', fs, nSeg(i), nSeg(i+1)-nSeg(i))';
            lenSeg = size(xi,1); % segment length

            % Kumar's approach to find between-class scatter
            xv = xi - ones( lenSeg ,1)*Xbc';
            T = T + xv'*xv;

            % find within-class scatter
            xj = xi - ones( lenSeg ,1)*Xjbc(:, j)';
            Wj{j} = Wj{j} + xj'*xj;

            disp([' computed segments ', num2str(nSeg(i)), '~', ...
              num2str(nSeg(i)+lenSeg-1), ' in class ', num2str(j)]);

        end

    end

    WjB{j} = Wj{j}/NJc(j);

```

```
end

% sum them up for all classes
Wj2 = cat(3,Wj{:});

% normalization (not necessary since both TB and WB are normalized by the
% same factor)
TB = T/Nc;
WB = sum(Wj2,3)/Nc;
```

codes/hlda2/main_solve.m

```
% Solves for the Optimal theta(s) for the case p=2, and displays the
% selected sub-space on the screen
% Then, computes the classification performances for the cases p=2, and p=5
% under different models
% Assumes n>4, and J<17
% Assumes that the variables have been setup properly
% Set ccomp = 0 if you dont want to solve for theta
%
% Programmar: Nagendra Kumar
% Modified by Zhenhao Ge, 2012-12-09

%% Initialization

global p NJ

% load setting from setup
% load('setup_equal2');
load('setup_20121209T181021');

% % swith back to old format
% x = cell2mat(x);
% Tx = cell2mat(Tx);
% classes = [0;cumsum(NJ)];
% Tclasses = [0;cumsum(TNJ)];

% find n,N,J,NJ,TNJ
[NJ,n] = cellfun(@size,x);
if range(n) ~= 0
    error('features of classes should have the same dimension!');
else
    n = n(1);
end
TNJ = cellfun(@length,Tx);
N = sum(NJ);
J = length(NJ);

% initialize theta
fs = 2*c+1;
theta = zeros(fs*n,fs*n,6);

% loglik in case(i,j), where i is case of subspace dimension (1 or 2), j is
% case of covariance matrix (1, 2, or 3)
m = zeros(2,3);

% specify color space and titil for each case
Xcol = [0 0 0; 0 0 1; 0 1 0; 0 1 1;
```

```
          1 0 0; 1 0 1; 1 1 0; 1 1 1];
strTitle = {'1: Projection using LDA';
            '2: Projection with HLDA: unequal diagonal S';
            '3: Projection with HLDA: unequal S'};

% option to compute theta
comp = 1;

%% subspace dimension 2

% subspace dimension
p = 2;

% compute transform matrix in each case
if comp == 1
    theta(:,:,1) = problem(TB,WB,1,options);
    theta(:,:,2) = problem(TB,WB,2,options,theta(:,:,1));
    theta(:,:,3) = problem(TB,WB,3,options,theta(:,:,1));
end

% figure(4), hold on,
% for j = 1:J
%     h = plot(x{j}(:,3),x{j}(:,4),'+');
%     set(h,'col',Xcol(j,:));
% end
% title('original data'), hold off;

% 2D plot for demonstration
Xt = cell(J,6);
close all
for i = 1:3

    figure(i), hold on,
    for j = 1:J
        % obtain the dimension-reduced feature
        xi = matola(x{j}', fs)';
        Xt{j,i} = xi*theta(:,1:p,i);
        % plot 2D data
        h = plot(Xt{j,i}(:,1),Xt{j,i}(:,2),'+');
        set(h,'col',Xcol(j,:));
    end
    title(strTitle{i}), hold off,

end

% compute the loglik and accuracy for each case
p2 = zeros(3,3);
for i = 1:3
    m(1,i) = loglik(theta(:,:,i),i);
    for j = 1:3
        p2(i,j) = test(theta(:,:,i),Tx,j);
    end
end

% display accuracy
disp('p2 = '); disp(p2);

%% subspace dimension 5
```

```
% subspace dimension
p = 5;

% compute transform matrix in each case
if comp == 1
    theta(:,:,4) = problem(TB,WB,1,options);
    theta(:,:,5) = problem(TB,WB,2,options,theta(:,:,2));
    theta(:,:,6) = problem(TB,WB,3,options,theta(:,:,3));
end

% compute the loglik and accuracy for each case
p5 = zeros(3,3);
for i = 1:3
    m(2,i) = loglik(theta(:,:,i),i);
    for j = 1:3
        p5(i,j) = test(theta(:,:,i+3),Tx,j);
    end
end

% display accuracy and loglik
disp('p5 = '); disp(p5);
disp('m = '); disp(m);
```

codes/hlda2/problem.m

```
function theta = problem(TB,WB,type,options,theta0)
% This is a implementation of LDA
% Usage: theta1 = problem1(TB,WB)
% Programmer: Nagendra Kumar

n = size(TB,1);

if type == 1 || nargin < 4

    [V,D] = eig(WB\TB);
%     [V,D] = eig(TB,WB,'qz'); % from ldatrace.m

    % simplified but identical approach
    [~,I] = sort(real(diag(D)')-1,'descend');
    theta0 = V(:,I);
    theta0 = real(theta0) + imag(theta0);
    t = det(theta0);
    if t < 0
        theta0(:,n) = -1*theta0(:,n);
    end

%     % original approach
%     [~,I] = sort(real(diag(D)')-1);
%     a = size(TB,1);
%     for i = a:-1:1
%         theta0(a-i+1,:) = V(:,I(i))';
%     end
%     theta0 = real(theta0) + imag(theta0);
%     t = det(theta0);
%     if t < 0
%         theta0(n,:) = -1*theta0(n,:);
%     end
```

```
%       theta0  = theta0 ';

end

tt  = fminsd( theta0 , options , type);
theta  = reshape( tt ,n,n);

% Following  normalizations  make the matrix  theta2  well  scaled , and do not
% affect  the  log- likelihood
theta  = theta /( diag( sqrt ( diag( theta * theta ') )));
theta  = theta /( det( theta )^(1/n));
```

codes/hlda2/fminsd.m

```
function  [xc,OPTIONS] = fminsd(xc,OPTIONS,type)
% Uses  steepest  descent  to  find  the minimum of  the  supplied  function .  Needs
% the  gradients  to  be  suplied  explicitly .
%
% Usage  :  fminsd(FUN,xc,OPTIONS,GRADFUN)
% FUN    :  String  argument - name of  the  function  to  be  optimized
% xc     :  Initial  Guess  of  the  value
% OPTIONS: Various  options  on  feedback  and  termination  confitions  follow
% the same  format  as MATLAB optimization  toolbox
% GRADFUN: String argument - function  that  computes  gradient  of  FUN
%
% Written  by  Nagendra  Kumar
% Modified  by  Zhenhao  Ge,  2012-12-09

xc  = xc(:);
f  =  loglik (xc,type);
nn  =  length (xc);
n  =  sqrt (nn);

g  =  gradient (xc,type);
if  OPTIONS(18) > 0
    step  = OPTIONS(18);
else
    step  =  0.01;
end

OLDX = xc;
OLDF = f;
OLDG = g;
nog  = norm(OLDG);

ng  = nog;

tt  =  reshape (xc,n,n);
e  =  0;

OPTIONS(10) = 1;  % F  cnt
OPTIONS(11) = 1;  % Gradient  Cnt
status  = -1;
improv  =  0;
disp ([f,  improv,  step,  det(tt)]);
cnt  =  1;
while  status  < 2
    if  (e < 1)
```

```
45          status = -1;
    end
47  e = 0;
    xc = OLDX - step*g;
49  f = loglik(xc,type);
    OPTIONS(10) = OPTIONS(10)+1;
51  if f > OLDF

53          step = step /2.0;

55          if (step*ng) < OPTIONS(2)
                disp(['step', sprintf('%5.5e', step*ng) '<', ...
57                  sprintf('%5.5e', OPTIONS(2))]);
                status = status + 1;
59              e = 1;
            end
61
            if OPTIONS(10) < 80
                status = -1;
            end
65
            if OPTIONS(14) < OPTIONS(10)
67              disp(['Max # of iterations exceeded', ...
                    sprintf('%7.0f', OPTIONS(10))]);
69              status = status + 1;
                e = 1;
71          end

73      else

75          improv = OLDF - f;
            g = gradient(xc,type);
77          OPTIONS(11) = OPTIONS(11)+1;
            ng = norm(g);
79      % Now a Heuristic formula for Step Size
        step = step *0.85*(1+0.6*( sum(OLDG.*g)/(nog*ng)));
81      cnt = cnt+1;
        if cnt > OPTIONS(1)
83          if OPTIONS(1) > 0
                tt = reshape(xc,n,n);
85              fprintf('%d %e %e %e %e\n', OPTIONS(10), ...
                    f, improv, step*ng, det(tt));
87              cnt = 1;
            end
89      end
        OLDX = xc;
91      OLDF = f;
        OLDG = g;
93
        nog = ng;
95      if (step*ng) < OPTIONS(2)
            disp(['step', sprintf('%5.5e', step*ng) '<', ...
97              sprintf('%5.5e', OPTIONS(2))]);
            status = status +1;
99          e = 1;
        end
101
        if improv < OPTIONS(3)*0.2
```

```
            disp ([ 'Df', sprintf (' %5.5e', improv) '<', ...
                sprintf (' %5.5e', OPTIONS(3))]);
            status = status +1;
            e = 1;
        end

        if ng < OPTIONS(4)
            disp ([ 'G', sprintf (' %5.5e', ng) '<', ...
                sprintf (' %5.5e', OPTIONS(4))]);
            status = status +1;
            e = 1;
        end

        if OPTIONS(10) < 80
            status = -1;
        end

        if OPTIONS(14) < OPTIONS(10)
            disp ([ 'max # of iterations exceeded ', ...
                sprintf ('%7.0f', OPTIONS(10))]);
            status = status +1;
            e = 1;
        end
    end
end
```

codes/hlda2/loglik.m

```
function f = loglik (tt, type)
% Function Le(theta|x) without the constant terms (input is theta)
% Assumes that the necesseary variables have been declared as global
% It scales the likelihood using the MDL criterion
% Programmer: Nagendra Kumar

global p NJ WB WjB TB

N = sum(NJ);
J = length (NJ);
tt = tt (:) ;
n = sqrt ( length (tt) );
theta = reshape (tt,n,n);
theta_p = theta (:,1: p);
theta_np = theta (:, p+1:n);
t1 = log (prod (diag (theta_np '* TB*theta_np)));

if type == 1
    t2 = log (prod (diag (theta_p '* WB*theta_p)));
    npar = J*p+(n-p)+(n*(n+1)/2);
elseif type == 2
    t2 = 0;
    for j = 1:J
        t2 = t2 + (NJ(j)/N) *...
            log (prod (diag (theta_p '* WjB{j}*theta_p)));
    end
    npar = 2*J*p + 2*(n-p) + n*(n-1)/2;
elseif type == 3
    t2 = 0;
    for j = 1:J
```

```
31            t2  = t2  + (NJ(j)/N) *  ...
                  log(det(theta_p'*WjB{j}*theta_p));
33      end
        npar  = J*p  + (n-p)  + (J-1)*p*(p+1)/2  + n  + (n*(n-1)/2);
35  end

37  f  =  0.5*(t1+t2)  -  log(det(theta))  + (npar/2)*(log(N)/N);
```

codes/hlda2/gradient.m

```
1   function  df  =  gradient(tt,type)
    % Computes  gradient  of  -Le(theta|x)  (input  is  theta)
3   % Assumes  that  the  necessary  variables  have  been  declared  as  global
    % Programmer:  Nagendra  Kumar
5
    global  p  NJ  WB  WjB  TB
7
    N = sum(NJ);
9   J  =  length(NJ);
    tt  =  tt(:);
11  n  =  sqrt(length(tt));
    theta  =  reshape(tt,n,n);
13  theta_p  =  theta(:,1:p);
    theta_np  =  theta(:,p+1:n);
15
    if  type  ==1
17      t1  =  WB*theta_p*diag(1./diag(theta_p'*WB*theta_p));
     elseif  type  ==2
19      t1  =  0;
        for  j  =  1:J
21          t1  =  t1  +  (NJ(j)/N)*WjB{j}*theta_p  *...
                  diag(1./diag(theta_p'*WjB{j}*theta_p));
23      end
     elseif  type  ==3
25      t1  =  0;
        for  j  =  1:J
27          t1  =  t1  +  (NJ(j)/N)*WjB{j}*theta_p/(theta_p'*WjB{j}*theta_p);
        end
29  end

31  t2  =  TB*theta_np*diag(1./diag(theta_np'*TB*theta_np));
    t3  =  [t1,t2]  -  inv(theta');
33  df  =  t3(:);
```

codes/hlda2/test.m

```
1   function  [accuracy, correct, cr]  =  test(theta,xx,type)
    % Program  for  testing  on  the  test  data
3   % Programmer:  Nagendra  Kumar

5   global  p  WB  WjB  Xjb

7   % find  frame  size
    fs  =  size(Xjb,1)/size(xx{1},2);
9   if  round(fs)  ~=  fs
        error('fs  is  not  integer!');
11  end
```

```
J = length(WjB);
theta_p = theta(:,1:p);
mu_hat = theta_p'*Xjb;

lgs = zeros(J,1);
S_hat = cell(J,1);
IS_hat = cell(J,1);
Xt = cell(J,1);
for j = 1:J

    if type == 1
        S_hat{j} = diag(diag(theta_p'*WB*theta_p));
    elseif type == 2
        S_hat{j} = theta_p'*WjB{j}*theta_p;
    elseif type == 3
        S_hat{j} = diag(diag(theta_p'*WjB{j}*theta_p));
    end

    lgs(j) = log(det(S_hat{j}));
    IS_hat{j} = inv(S_hat{j});

    xxi = matola(xx{j}', fs)';
    Xt{j} = xxi*theta_p;

end

TNJ = cellfun(@length,Xt);
TN = sum(TNJ);
Dt = cell(J,J);
lgl = cell(J,J);
classif = cell(J,1);
correct = zeros(J,1);
for j = 1:J % jth model

    lgl{j} = zeros(TNJ(j),1);
    for i = 1:J % ith class of data
        Dt{j,i} = Xt{i} - ones(TNJ(i),1)*mu_hat(:,j)';
        Dt{j,i} = (Dt{j,i}*IS_hat{j}).*Dt{j,i};
        Dt{j,i} = sum(Dt{j,i},2);
        lgl{j,i} = -Dt{j,i} - lgs(j);
    end

end

for i = 1:J
    lgl2 = cell2mat(lgl(:,i)');
    [~, classif{i}] = max(lgl2,[],2);
    correct(i) = sum(classif{i}==i);
end

accuracy = sum(correct)/TN;
cr = correct./TNJ;
```

codes/hlda2/main_hlda.m

```
% Main function of transform original feature to HLDA feature using
% Steepest Descent (SD) optmization with various assumptions in class
% covaraince and option settings in optimization
```

```
%
% Author: Roger Ge, roger.ge@inin.com
% Date created: 2012-12-03

global p NJ

% load setting from setup
load('data_unequal3_200-201-199-200','x');

% set up numerical optimization option
options = foptions;
% Provide a feedback every options(1) iterations
options(1) = 200;
%options(1) = 0;
% Minimum Step Size
options(2) = 1e-8;
% Minimum improvement at every iteration
options(3) = 1e-8;
options(6) = 1;
options(9) = 0;
% options(9) = 1;
options(13) = 0;
% Maximum number of iterations
options(14) = 1000000;
options(16) = 1e-11;
options(17) = 1e-3;
% Initial guess of step-size
options(18) = 0.001;
save('hlda_opts','options');

% find n,J,NJ
J = length(x);
[NJ,n] = cellfun(@size,x);
if range(n) ~= 0
    error('features of classes should have the same dimension!');
else
    n = n(1);
end

p = 2;
c = 1;
type = 3;
theta = hlda(x,c,type,options);

Xt = cell(J,1);
for j = 1:J
    Xt{j} = x{j}*theta(:,1:p);
end
```

codes/hlda2/hlda.m

```
function theta = hlda(x,c,type,options,theta0)
% Find the HLDA transform matrix using the original feature
%
% Reference: Thesis from Nagendra Kumar
%
% Author: Roger Ge, roger@inin.com
% Date created: 2012-12-03
```

```
 global  WB WjB TB Xjb

[WB,WjB,TB,Xjb] =   initialize2  (x,c);

n = size(TB,1);

if type == 1 || nargin < 5

    [V,D] = eig(WB\TB);
    [~, I] = sort(real(diag(D)'),-1,'descend');
    theta0 = V(:, I);
    theta0 = real(theta0) + imag(theta0);
    t = det(theta0);
    if t < 0
        theta0(:, n) = -1*theta0(:, n);
    end

    % find theta using steepest descent optimization
    tt = fminsd(theta0, options,1);
    theta = reshape(tt,n,n);

    % normalization of theta
    theta = theta/(diag(sqrt(diag(theta*theta'))));
    theta = theta/(det(theta)^(1/n));

end

% type 2 and 3
if type == 2 || type == 3

    % find theta using steepest descent optimization
    if exist('theta','var')
        theta0 = theta;
    end
    tt = fminsd(theta0, options,type);
    theta = reshape(tt,n,n);

    % normalization of theta
    theta = theta/(diag(sqrt(diag(theta*theta'))));
    theta = theta/(det(theta)^(1/n));

end
```

VITA

VITA

Zhenhao Ge is a PhD student and Research Assistant at department of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana. He is currently working with Prof. Mark J.T. Smith on speech recognition and accent detection based on statistical modeling and pattern recognition. His interested area involves speech, image and video processing, computer vision, machine learning and information retrieval, etc.