KERNEL EIGENSPACE-BASED MLLR ADAPTATION

by

HSIAO WEND HUU, ROGER

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science

November 2004, Hong Kong

Copyright © by Hsiao Wend Huu, Roger 2004
Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

HSIAO WEND HUU, ROGER
KERNEL EIGENSPACE-BASED MLLR ADAPTATION

by

HSIAO WEND HUU, ROGER

This is to certify that I have examined the above M.Phil. thesis and have found that it is complete and satisfactory in all respects, and that any and all revisions required by the thesis examination committee have been made.

DR. BRIAN MAK, THESIS SUPERVISOR

PROF. LIONEL NI, HEAD OF DEPARTMENT

Department of Computer Science

26 November 2004
ACKNOWLEDGMENTS

I would like to express my sincere thankfulness to Dr. Brian Mak for his supervision throughout my MPhil study. I would also like to thank Dr. James Kwok and Dr. Tan Lee for being my panel and giving valuable suggestions for improving my work.

I am pleased to be a part of LASTRE group guided by Dr. Brian Mak and Dr. Manhung Siu. Through different research projects, I gained research experience and have exposure to different areas of speech and language technologies.

Finally, I would like express my gratitude to my colleagues including Wilson Tam, Arthur Chan, Jimmy Wong, Ivan Chan, Franco Chong, Mimi Ng, Karen Leung, Jacqueline Lo, Simon Ho, Chak-Fai Li, Siu-Man Chan, Wing-Hei Au, Harry Lai, Ka-Keung Wong, Tim Ng and Jeff Au Yeung. I learned a lot from them in the past 2 years.
TABLE OF CONTENTS

Title Page i
Authorization Page ii
Signature Page iii
Acknowledgments iv
Table of Contents v
List of Figures viii
List of Tables x
Abstract xi

Chapter 1 Introduction 1
  1.1 Background 1
  1.2 Outline of the Thesis 3

Chapter 2 Rapid Speaker Adaptation 5
  2.1 Speaker-Independent and Speaker-Dependent Modeling 5
  2.2 Eigenvoice (EV) Rapid Speaker Adaptation 6
    2.2.1 Principal component analysis (PCA) 7
    2.2.2 Parameter spaces of EV 8
    2.2.3 Eigenvoice speaker adaptation 9
    2.2.4 EV on multiple-mixture HMM 11
    2.2.5 Robust EV 12
  2.3 Maximum Likelihood Linear Regression (MLLR) Speaker Adaptation 12
    2.3.1 Regression class 13
    2.3.2 Maximum likelihood estimation framework of MLLR 14
  2.4 Eigenspace-based MLLR (EMLLR) Speaker Adaptation 16
    2.4.1 Parameter spaces of EMLLR 17
    2.4.2 Mechanism of EMLLR 17
    2.4.3 Robust EMLLR 20
### Chapter 3  Improving Eigenvoice-based Adaptation by Kernel PCA

3.1 Kernel Methods
   - 3.1.1 Kernel principal component analysis (KPCA)

3.2 Kernel Eigenvoice (KEV) Speaker Adaptation
   - 3.2.1 Parameter spaces of KEV
   - 3.2.2 Mechanism of KEV
   - 3.2.3 Composite kernel
   - 3.2.4 KEV adaptation

3.3 Speedup of KEV: Embedded KEV Adaptation (eKEV)
   - 3.3.1 eKEV adaptation

3.4 Experimental Setup: Resource Management
   - 3.4.1 Experimental evaluation of EV
   - 3.4.2 Experimental evaluation of MLLR
   - 3.4.3 Experimental evaluation of EMLLR
   - 3.4.4 Experimental evaluation of eKEV
   - 3.4.5 Comparison among EV, MLLR, EMLLR and eKEV

### Chapter 4  Kernel Eigenspace-based MLLR (KEMLLR) Adaptation

4.1 Overview of KEMLLR

4.2 Parameter Spaces of KEMLLR

4.3 KEMLLR Adaptation
   - 4.3.1 Kernel eigenmatrices in the feature space
   - 4.3.2 Composite kernel
   - 4.3.3 Kernel evaluation
   - 4.3.4 Gradient of Gaussian means
   - 4.3.5 ML estimation of eigenmatrix weights by the quasi-Newton BFGS method
   - 4.3.6 Robust KEMLLR

### Chapter 5  Experimental Evaluation

5.1 Experimental Procedure

5.2 Finding the Kernel Parameter $\beta$

5.3 Effectiveness of BFGS quasi-Newton Method

5.4 Performing KPCA(PCA) Separately or Simultaneously on Different Regression Classes
5.5 Number of Eigenmatrices and Regression Classes 51
5.6 Comparison among Various Adaptation Methods 53
5.7 Analysis of the Eigenmatrices 55
5.8 Significance Tests 67
5.9 Time Complexity Analysis of Online Computation 67
  5.9.1 Time complexity of EV 68
  5.9.2 Time complexity of EMLLR 69
  5.9.3 Time complexity of MLLR 70
  5.9.4 Time complexity of KEV 70
  5.9.5 Time complexity of eKEV 71
  5.9.6 Time complexity of KEMLLR 72
  5.9.7 Summary of Time Complexity Analysis 72

Chapter 6 Conclusion and Future Work 74
  6.1 Summary 74
  6.2 Conclusion 74
  6.3 Contributions 75
  6.4 Future work 76

References 77

Appendix A Proofs for KPCA 82
  A.1 Proof of centering of covariance matrix 82
  A.2 Proof of the normalizing factor used in KPCA 82

Appendix B Implementation issues 84
  B.1 Introduction to BLAS and LAPACK 84
  B.2 Experimental Evaluation 85

Appendix C Significance Tests 86
# LIST OF FIGURES

1.1 Adaptation Methods. 2  
2.1 Overview of EV. 6  
2.2 MLLR regression classes. 13  
2.3 MLLR regression tree. 14  
2.4 Overview of EMLLR. 18  
3.1 An example from "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". 21  
3.2 Overview of KEV. 25  
3.3 A preimage of a supervector in the feature space. 30  
3.4 Performance of EV with a single-mixture HMM. 35  
3.5 Performance of EMLLR with a single-mixture HMM. 37  
3.6 Performance of eKEV with a single-mixture HMM. 39  
5.1 Finding a suitable value of $\beta$ for the Gaussian kernel in KEMLLR adaptation. 49  
5.2 Performance of EMLLR adaptation with a 10-mixture HMM. 52  
5.3 Performance of KEMLLR adaptation with a 10-mixture HMM. 53  
5.4 Performance of eKEV adaptation with a 10-mixture HMM. 54  
5.5 Performance of KEMLLR adaptation with a single-mixture HMM using different numbers of eigenmatrices. 55  
5.6 Scatter plot of EMLLR for adults using 40 eigenmatrices. 57  
5.7 Scatter plot of KEMLLR for adults using 40 eigenmatrices. 57  
5.8 Scatter plot of EMLLR for adults using 70 eigenmatrices. 58  
5.9 Scatter plot of KEMLLR for adults using 70 eigenmatrices. 58  
5.10 Scatter plot of EMLLR for adults using 100 eigenmatrices. 59  
5.11 Scatter plot of KEMLLR for adults using 100 eigenmatrices. 59  
5.12 Scatter plot of EMLLR for adults using 130 eigenmatrices. 60  
5.13 Scatter plot of KEMLLR for adults using 130 eigenmatrices. 60  
5.14 Scatter plot of EMLLR for adults using 163 eigenmatrices. 61  
5.15 Scatter plot of KEMLLR for adults using 163 eigenmatrices. 61  
5.16 Scatter plot of EMLLR for all speakers using 40 eigenmatrices. 62  
5.17 Scatter plot of KEMLLR for all speakers using 40 eigenmatrices. 62  
5.18 Scatter plot of EMLLR for all speakers using 70 eigenmatrices. 63
5.19 Scatter plot of KEMLLR for all speakers using 70 eigenmatrices. 63
5.20 Scatter plot of EMLLR for all speakers using 100 eigenmatrices. 64
5.21 Scatter plot of KEMLLR for all speakers using 100 eigenmatrices. 64
5.22 Scatter plot of EMLLR for all speakers using 130 eigenmatrices. 65
5.23 Scatter plot of KEMLLR for all speakers using 130 eigenmatrices. 65
5.24 Scatter plot of EMLLR for all speakers using 163 eigenmatrices. 66
5.25 Scatter plot of KEMLLR for all speakers using 163 eigenmatrices. 66
LIST OF TABLES

3.1 RM1: MLLR adaptation results with a single-mixture HMM. 36
3.2 Comparison of the best performance of different adaptation methods on a single-mixture HMM using 5 seconds of adaptation data. 39
3.3 Comparison of the best performance of different adaptation methods on a single-mixture HMM using 10 seconds of adaptation data. 40
3.4 Comparison of the best performance of different adaptation methods with a 10-mixture HMM using 5 seconds adaptation data. 40
3.5 Comparison of the best performance of different adaptation methods with a 10-mixture HMM using 10 seconds adaptation data. 41
5.1 EMLLR results obtained by using BFGS and by solving a system of linear equations. 50
5.2 Performing KPCA(PCA) separately or simultaneously on 2 regression classes. 51
5.3 Performance of the SI model, MLLR, EMLLR, eKEV and KEMLLR adaptation using a 10-mixture HMM. 54
5.4 Adaptation performance of the SI model, MLLR, EMLLR, eKEV and KEMLLR adaptation with a single-mixture HMM. 55
5.5 Time complexities of EVP, MLLR and EMLLR. 73
5.6 Time complexities of KEV, eKEV and KEMLLR. 73
B.1 Averaged time of computing multiplication of two $1024 \times 1024$ matrices. 85
C.1 Significance tests on 5-second adaptation data using a 10-mixture HMM. 87
C.2 Significance tests on 10-second adaptation data using a 10-mixture HMM. 88
KERNEL EIGENSPACE-BASED MLLR
ADAPTATION

by

HSIAO WEND HUU, ROGER

Department of Computer Science
The Hong Kong University of Science and Technology

ABSTRACT

Kernel methods have been applied to improve the performance of existing eigenvoice-based adaptation methods. Several adaptation methods including kernel eigenvoice adaptation (KEV) and embedded kernel eigenvoice adaptation (eKEV) report promising results. The basic idea of these kernel-based adaptation methods is to exploit possible nonlinearity among different speakers.

In this thesis, a variant of eigenspace-based maximum likelihood linear regression (EMLLR) adaptation, named kernel eigenspace-based MLLR adaptation (KEMLLR), is proposed. It adopts kernel methods to map the MLLR transformation matrices to a kernel-induced high dimensional space and kernel principal component analysis (KPCA) is used to derive a set of kernel eigenmatrices in the feature space. A new speaker is then represented by a linear combination of the leading kernel eigenmatrices in the feature space, and the eigenmatrix weights are optimized by using BFGS quasi-Newton algorithm.

In the Resource Management adaptation task using 10-mixture monophone hidden Markov models, KEMLLR gives encouraging results in rapid speaker adaptation. It is found that when only 5s of adaptation speech are available, EMLLR successfully reduces the word error rate (WER) by 7.82%, and KEMLLR can reduce the WER by 11.4%. When 10s of adaptation speech are provided,
MLLR using full transformation (MLLR-F) becomes effective and matches the performance of KEMLLR. EMLLR, eKEV and MLLR using diagonal transformation (MLLR-D) do not perform as well as KEMLLR in the experiments.

The time complexities of various adaptation methods including EV, MLLR, EMLLR, KEV, eKEV, and KEMLLR are also analyzed under some mild assumptions. Comparisons are made and the analysis helps understanding the efficiency of various methods.
CHAPTER 1

INTRODUCTION

1.1 Background

Speaker adaptation techniques have developed rapidly in the past few years. With a limited amount of speaker specific information, speaker adaptation tries to improve the recognition accuracy of the adapting speaker. A successful speaker adaptation method can benefit many applications including various voice-control appliances, computer aided language learning, dictation software and so on.

Existing speaker adaptation methods can be roughly divided into two types: feature based and model-based methods. Vocal-tract normalization (VTLN) [15] is a feature based adaptation method which eliminates speaker variations caused by different vocal tract lengths. The features are normalized so that the effect of the vocal tract is removed. Some feature-based adaptation methods also consider variations due to other speaker characteristics. In [40], articulatory feature, which contains information about a sound such as voicing and labial, is incorporated into a multiple stream architecture for speaker adaptation.

Model-based speaker adaptation techniques can be further divided into three categories: Bayesian-based, transformation-based and eigenspace-based methods. They are shown in figure 1.1. These methods require the adapting speaker to provide certain amount of speech, and they use the information to improve the recognition performance of the adapting speaker. Due to the fact that clients usually do not want to spend too much time on training the system, it is desirable to have an adaptation method which needs very small amount of adaptation data. This leads to the study of rapid speaker adaptation. To be more precised, in this thesis, rapid speaker adaptation means adaptation with less than 10 seconds of adaptation data. It turns out that many conventional adaptation methods break down when there are less than 10 seconds of adaptation data.

MAP adaptation [16] [17] [30] is a Bayesian-based method. It takes advantage of prior knowledge such that the model parameters are adjusted in a way that is
Speaker Adaptation Methods

Bayesian-based
Transformation-based
Eigenspace-based

Without Kernel Method
MAP
MLLR
Eigenvoice

Eigenspace-based MLLR

With Kernel Method
Kernel Eigenvoice
Embedded Kernel Eigenvoice

Figure 1.1: Adaptation Methods.

guided by prior information. [22] showed that MAP estimate is a weighted average between the mode of the prior density and the ML estimate. MAP adaptation method has some limitations and in general, it converges slowly and needs a relatively large amount of adaptation data, compared with other transformation-based or eigenspace-based approaches.

MLLR adaptation [33] is one of the most commonly used transformation-based adaptation methods and it is described in more detail in chapter 2. Basically, it adopts a block diagonal or full transformation with regression classes to transform the hidden Markov model (HMM) parameters (mean vectors and/or covariance matrices). The number of parameters can be adjusted by choosing the number of regression classes and the structure of the transformation, and hence, MLLR is very flexible. Although in practice, MLLR needs less data compared with MAP, in cases where there is only a very small amount of data, MLLR may still not show a good performance.

Eigenspace-based adaptation methods including eigenvoice (EV) [24] [26] [25]
and eigenspace-based MLLR (EMLLR) [11] [13] [12] [3] which target rapid speaker adaptation, where there is only a very limited amount of adaptation data. The reason for having this limitation is, as mentioned in [27], because clients do not want to spend time training the system. EV and EMLLR adaptation basically perform principal component analysis (PCA) on training speakers to obtain some eigenvoices, and a new speaker is then expressed as a linear combination of those eigenvoices. The linear weights are determined by the maximum likelihood (ML) approach. More details about eigenspace-based approaches are available in chapter 2.

In 2004, an improvement of existing eigenspace-based eigenvoice (EV) adaptation is proposed [28][37] named Kernel Eigenvoice (KEV) adaptation. It exploits possible nonlinearity among each training speaker by using kernel PCA (KPCA). KEV, and its variant eKEV [36], achieves good adaptation performance and successfully capture speaker information in the experiments.

In this thesis, KPCA will be applied on EMLLR and the resulting adaptation method is named kernel eigenspace-based MLLR (KEMLLR) adaptation. The design of KEMLLR targets at achieving good rapid speaker adaptation performance when using complex acoustic models, and hopefully, KEMLLR is able to improve various real world applications which need fast adaptation. The basic idea is to map the speaker’s MLLR transformation supervectors to a high dimensional space via some nonlinear map, and then apply PCA to derive the eigenmatrices in the feature space. One major challenge in KEMLLR adaptation is to preserve the row information in the transformation supervectors, which will generally be lost during the mapping to the kernel-induced feature space. In chapter 4, the details of KEMLLR is given.

1.2 Outline of the Thesis

In chapter 2, the difference between speaker independent and speaker dependent modeling is discussed. Background information about various existing adaptation methods including EV, MLLR and EMLLR are reviewed.

Chapter 3 is mainly about the existing eigenvoice-based adaptation methods using KPCA. This includes KEV and its speedup version eKEV. Some simple
experiments are conducted to compare various adaptation methods.

In chapter 4, the KEMLLR algorithm is presented. Details about how KPCA can be applied to improve the EMLLR adaptation are mentioned. Some challenges and proposed solutions are discussed.

Chapter 5 is about the experimental evaluation of various adaptation methods. Comparison will be made and the effectiveness of KEMLLR is investigated. The conclusion and the future work are discussed in the last chapter.
CHAPTER 2

RAPID SPEAKER ADAPTATION

2.1 Speaker-Independent and Speaker-Dependent Modeling

In speech recognition, the spoken sound is modeled by acoustic models. There are many possible choices for the acoustic models including neural network [19], Gaussian mixture model[1], dynamic time warping [44], or the most commonly used; hidden Markov model (HMM) [31]. When applying HMM in a speech recognition task, one has to provide speech samples as training data and apply the EM algorithm to estimate the model parameters, so that the trained HMM can best explain the provided training data.

Depending on the nature of the task, data from various speakers can be supplied to train the HMMs or we can use the data of only one speaker. In the first case, the resulting model is known as the speaker-independent (SI) model, while the latter is known as the speaker-dependent (SD) model.

The advantage of an SI model is that it can perform fairly well for different speakers since an SI model can handle speaker variations. For example, different speakers may pronounce the word “tomato” in different ways. Such variation may be due to accent, age, gender, rate of speech, and other factors. In order to handle these variations, a large amount of training data from various speakers is required to capture this information in the model. This is one of the major drawbacks of SI modeling.

In contrast to SI modeling, SD modeling targets only one specific speaker and the developed model can achieve very good accuracy for that speaker. It is useful since for some applications, like smart home or dictation software, they only have to serve a few specified users. Instead of gathering a huge amount of data to develop an SI model, it is more reasonable to develop an SD model. On the one hand, recognition can be performed more accurately; on the other hand, less data are required to build it. However, a disadvantage is that, the SD model
still requires a relatively large amount of data from the target speaker, which is sometimes infeasible.

The pros and cons of SI and SD modeling raise an important question in speech technology: Can we achieve the accuracy of the SD model for the target speaker given a minimal amount of speaker specific information? This motivates the development of various speaker adaptation (SA) techniques.

In this chapter, some popular model based speaker adaptation methods including eigenvoice, MLLR, eigenspace-based MLLR are reviewed. Some simple experiments are conducted to evaluate various adaptation methods and they are available in the next chapter.

### 2.2 Eigenvoice (EV) Rapid Speaker Adaptation

This section introduces the eigenvoice speaker adaptation technique which was developed by Kuhn et. al. in 1998 [24]. EV adaptation targets rapid speaker adaptation which tries to achieve good performance with a minimal amount of adaptation data. Figure 2.1 shows the overview of EV adaptation.

![Figure 2.1: Overview of EV.](image-url)
Eigenvoice speaker adaptation (EV) assumes that a speaker model is a linear sum of the training speaker models. Procedurally, the EV approach first develops an SD model for each training speaker, then given a new speaker and some adaptation data, the ML approach is applied to determine the linear weights so we can obtain a speaker model for the new speaker. This approach assumes that there are enough training speakers so a good model is likely to exist in the search space. However, as the number of training speakers increases, the number of parameters increases as well since there are more weights. To tackle this problem, EV adopts principal component analysis (PCA) to extract principal components, that are, the eigenvoices. As a result, the first few eigenvoices may already contain very rich speaker information and a new speaker can be represented by a weighted sum of eigenvoices.

2.2.1 Principal component analysis (PCA)

Principal component analysis (PCA) is one of the most widely used techniques for data analysis and dimension reduction. It applies linear methods to project high dimensional data to a low dimensional space, and the projection is found by minimizing the reconstruction error of the data. On the one hand, the classifier can be built in low dimensional space, so classification can be performed more efficiently. On the other hand, the projections provide important information about the structure of the data.

In principle, PCA searches for a set of basis vectors \( e = \{ e_1, e_2, \ldots \} \) which minimizes the following function [14],

\[
J_d = \sum_{k=1}^{N} \| (\bar{x} + \sum_{i=1}^{d} a_{ki} e_i) - x_k \|^2 ,
\]

(2.1)

where \( x_k \in \mathbb{R}^d \) is the \( k \)th data point and \( \bar{x} \) is the center of all \( x \). Minimizing \( J_d \) means that we would like to find a set of \( d \) basis vectors \( \{ e_1, e_2, \ldots, e_d \} \) that projecting \( x_1, \ldots, x_k \) on these basis vectors gives minimal reconstruction error. \( \{ e_1, \ldots, e_d \} \) are known as the principal components and any new data point is assumed to be a linear combination of these principal components. If \( d \) is smaller than the dimension of \( x \), dimension reduction is achieved since any new data point is projected onto a subspace.
PCA finds many applications in different areas. In speech recognition, PCA is applied to feature extraction, data decorrelation, robustness, speaker adaptation and so on. In [32], PCA is applied to feature extraction, in which, the filterbank coefficients are obtained by using PCA on the FFT spectrum of the training data, in order to improve the robustness under a noisy environment.

For the EV adaptation [24], [26], [11], PCA is applied to extract some principal components known as eigenvoices. Speaker characteristics is modeled by assuming that a new speaker is a linear combination of eigenvoices found by PCA and a set of training speakers. The application of PCA to speaker identification is also explored in [45], and it is based on a similar technique introduced in [24] and [26].

### 2.2.2 Parameter spaces of EV

Several parameter spaces are introduced in the EV-based approach.

- **Observation space**
  
  This is the acoustic feature space. The mel-frequency cepstral coefficient (MFCC) and the normalized energy from each speech frame are used as the acoustic features in this thesis. It is a sequence of 13-dimensional vectors. For some other tasks, delta MFCC and delta delta MFCC may be applied, and in such a case, the acoustic feature space would become 39-dimensional.

- **Supervector space**

  A supervector is formed by concatenating the means in the HMM. For the EV approach, PCA is applied on the supervector space. The dimension of a supervector can be calculated by the following formula:

  $\dim(\text{supervector space}) = \text{no. of HMMs} \times \text{no. of states per HMM} \times \text{no. of mixtures per state} \times \dim(\text{observation space}).$  

- **Eigenspace**

  This is the space obtained after performing PCA on the supervector space. Its dimension is equal to the number of eigenvoices in use. In the EV
approach, the dimension of this space is usually much smaller than the dimension of the supervector space.

### 2.2.3 Eigenvoice speaker adaptation

This section covers the mechanism of the EV adaptation. The first step of the EV adaptation is to train many SD models to form the supervector space. This can be done by recruiting many speakers and collecting a large amount of data from the training speakers. With the collected data, an SD model is trained for each speaker and an SI model is trained with all available data.

With the SD models, the mean vectors in the model can be extracted and concatenated to form a supervector. Note that the same modeling unit of all SD models must have the same topology. Although an HMM may consist of mean vectors, covariance matrices, mixture weights and transition probabilities, the EV approach only adjusts the means due to the linear combination assumption. Procedurally, the EV approach can also be applied to covariance matrices, but in such a case, the linear assumption may be violated.

In general, an HMM state may consist of multiple Gaussians, but for simplicity, the formulas below would assume a single Gaussian per state. Suppose there are only two modeling units, A and B, and each modeling unit is modeled by a strictly left-to-right HMM with two states only, and the observation space is 2-dimensional, then

HMM A: $\mu_1^A = [a_{11}, a_{12}]'$, $\mu_2^A = [a_{21}, a_{22}]'$

HMM B: $\mu_1^B = [b_{11}, b_{12}]'$, $\mu_2^B = [b_{21}, b_{22}]'$,

where $\mu_i^A$ denotes the mean vector of state $i$ in the A modeling unit, and $\mu_i^B$ denotes the mean vector of state $i$ in the B modeling unit. The resulting supervector is

$$[(\mu_1^A)', (\mu_2^A)', (\mu_1^B)', (\mu_2^B)']' = [a_{11}a_{12}a_{21}a_{22}b_{11}b_{12}b_{21}b_{22}]'$$.

By gathering $N$ speakers, where $N$ should be large, we can obtain $N$ supervectors. We may compute the covariance matrix of those supervectors, so we can perform PCA by applying eigendecomposition on the covariance matrix.
Eigendecomposition basically tries to solve

\[ C\nu = \lambda \nu \]
\[ \Rightarrow (C - \lambda I)\nu = 0 \]
\[ \Rightarrow |C - \lambda I| = 0 . \] (2.3)

After solving equation 2.3, we can pick \( M (\leq N) \) eigenvectors, \( e_m \) where \( m = 1, \ldots, M \) according to the descending order of the eigenvalues and those eigenvectors are called eigenvoices.

The new speaker’s supervector, \( s^{(ev)} \), is represented by a weighted sum of eigenvoices

\[ s^{(ev)} - e_0 = \sum_{m=1}^{M} w_m e_m, \]
\[ s^{(ev)} = \sum_{m=0}^{M} w_m e_m, \] (2.4)

where \( e_0 \) is the mean of all supervectors and \( w_m \) is the weight of the \( m \)-th eigenvoice. \( w_0 \) is always one in this case. The weights are usually estimated by the ML approach with the provided adaptation data. Since the EV approach only adjusts the mean vectors, the covariance matrices and the transition probabilities are copied from the SI model.

To maximize the likelihood of the adaptation data, we have to minimize the following auxiliary function,

\[ Q_b(w) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) (o_t - \mu_r(w))^\top C_r^{-1} (o_t - \mu_r(w)) , \] (2.5)

where \( R \) is the number of Gaussians in the model; \( \mu_r \) and \( C_r \) are the mean and the covariance of the \( r \)-th Gaussian respectively; \( o_t \) is the observation at time \( t \); \( \gamma_t(r) \) is the posterior probability of being at \( r \)-th Gaussian at time \( t \) given observation \( o_t \), and \( T \) is the length of the observation.

Since the new speaker’s supervector has the form of equation 2.4, The mean of the \( r \)-th Gaussian is given by,

\[ \hat{\mu}_r = \sum_{m=0}^{M} w_m e_{mr} . \] (2.6)
Hence, equation 2.5 can be rewritten as,

$$Q_b(w) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(o_t - \sum_{m=0}^{M} w_m e_{mr})^T C_r^{-1}(o_t - \sum_{m=0}^{M} w_m e_{mr}).$$  \hspace{1cm} (2.7)$$

Differentiate equation 2.7 w.r.t. each eigenvoice weight and set each derivative zero, \(\frac{\partial Q_b}{\partial w_k} = 0\), we obtain

$$\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)e_{kr}^T C_r^{-1}(o_t - e_{0r}) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) \left[ \sum_{m=0}^{M} w_m e_{kr}^T C_r^{-1} e_{mr} \right].$$  \hspace{1cm} (2.8)$$

Thus, the weights can be determined by solving equation 2.8 which represents a system of \(M\) linear equations with \(M\) variables.

PCA can also be performed by applying eigendecomposition on the correlation matrix of supervectors instead of the covariance matrix. This amendment can prevent variables with large scalar values from dominating the PCA. The procedure of using a correlation matrix on the EV approach is the same except that equation 2.8 becomes

$$\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)e_{kr}^T C_r^{-1}(o_t - e_{0r}) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) \left[ \sum_{m=0}^{M} w_m e_{kr}^T C_r^{-1} e_{mr} \right],$$  \hspace{1cm} (2.9)$$

where \(e_{mrk} = \sigma_{rk} e_{mr}\), and \(\sigma_{rk}^2\) is the variance of \(k\)-th component of the eigenvoice vector in the supervector space.

### 2.2.4 EV on multiple-mixture HMM

In the case where a multiple-mixture model is used, one has to handle the ordering of the mean vectors in the supervectors. Since there is no specific order for the Gaussians in the states of each SD model, inconsistent orderings across different SD models will lead to different results. A remedy to this problem is proposed in [8], in which, MLLR is applied to the SI model to create the speaker adapted (SA) models, and the SD models are replaced by the SA models. The advantage of this solution is that after a linear transformation, the order of the mean vectors in a vectorized SA model is the same as the vectorized SI model. Hence, the problem is solved and we can also take advantage of using the mixture weights, covariance matrices and the transition probabilities from the SI model in the newly adapted model. The disadvantage of this approach is that an SA model may not contain as much speaker information as an SD model.
2.2.5 Robust EV

As the amount of adaptation data is very small, the estimate found by EV may not be reliable. A parameter smoothing method is applied as in [21] where an adapted mean, $\mu_r^{(rev)}$ is an interpolation between the mean vector from the SI model, $\mu_r^{(SI)}$, and the mean vector found by the EV approach. The definition is as follows:

$$
\mu_r^{(rev)} = w_0\mu_r^{(SI)} + (1 - w_0)(e_0 + \sum_{m=1}^{M} w_m e_{mr}) \tag{2.10}
$$

where $0.0 \leq w_0 \leq 1.0$ and is determined by ML and gradient ascent approach. This interpolation method is similar to the moment interpolation algorithm proposed in [10], but instead of interpolating the expected log likelihood, interpolation is performed directly on the mean vectors in our case. For the EV adaptation, eigenvoice weights, $w_i$ where $i = 1, \ldots, M$, are first determined by solving the system of linear equations in equation 2.8, and then $w_0$ can be calculated by,

$$
w_0 = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) (o_t - \mu_r^{(ev)}) C_r^{-1} (\mu_r^{(ev)} - \mu_r^{(SI)})}{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) (\mu_r^{(ev)} - \mu_r^{(SI)}) C_r^{-1} (\mu_r^{(ev)} - \mu_r^{(SI)})} \tag{2.11}
$$

where $\mu_r^{(ev)}$ is the $r$th mean vector of speaker adapted model estimated by the EV approach, and $\mu_r^{(SI)}$ is the $r$th mean vector of the SI model. This formula is obtained by setting $\frac{\partial Q_b}{\partial w_0} = 0$. As shown in [21], this parameter smoothing can improve various eigenspace-based adaptation methods.

2.3 Maximum Likelihood Linear Regression (MLLR) Speaker Adaptation

In general, the amount of adaptation data is limited, so it would be impossible to train an SD model. To relieve this problem, MLLR, instead of determining all the parameters in an HMM, suggests a set of linear regression transformation for the Gaussians in an HMM, so that the likelihood of the adaptation data would be maximized. Specifically, after transformation, a Gaussian mean vector, $\mu_i$ of an HMM becomes

$$
\hat{\mu}_i = A\mu_i + b \tag{2.12}
$$
where $A$ is a transformation matrix and $b$ is a bias. MLLR determines $A$ and $b$ by maximizing the likelihood of the adaptation data. The number of free parameters is much less than a speaker-dependent model. As a result, even with a limited amount of adaptation data, MLLR can provide a speaker adapted (SA) model which has a good performance for that speaker.

2.3.1 Regression class

MLLR adaptation usually groups the Gaussians in an HMM into several groups known as regression classes. The Gaussians in the same regression class share a linear transformation. An underlying assumption is that similar Gaussians represent similar acoustic characteristics and such a relationship would remain after adaptation. This is shown in figure 2.2. The definition of similar Gaussians varies for different applications, but in practice, k-mean clustering on mean vectors is one of the most commonly used methods for deciding the membership of regression classes. With regression classes, equation 2.12 becomes

$$\hat{\mu}_i = A_c \mu_i + b_c ,$$

(2.13)

where $A_c$ and $b_c$ are the transformation matrix and the bias of regression class $c$ respectively.

![Figure 2.2: MLLR regression classes.](image)

The choice of the number of regression classes depends on the amount of available adaptation data. More regression classes implies that there is a greater
number of free parameters (since there are more regression matrices and biases), and they allow finer adjustments to the HMM parameters. However, when there is a lack of adaptation data, it would be desirable to reduce the number of regression classes so that there are fewer free parameters, and hence, the transformations can be more reliably estimated.

A commonly used scheme to achieve that is to build a regression tree as shown in Figure 2.3. Depending on the available amount of adaptation data, the regression tree would decide the groupings of the regression classes and determine how many regression matrices would be used for adaptation. This regression tree can be constructed by simply performing hierarchical clustering on the Gaussian mean vectors.

![Figure 2.3: MLLR regression tree.](image)

### 2.3.2 Maximum likelihood estimation framework of MLLR

As mentioned before, the regression matrices and biases are determined by the maximum likelihood estimation (ML) approach. This can be done by minimizing
the following auxiliary function, known as the $Q_b$ function,

$$Q_b = \sum_{s=1}^{S} \sum_{m=1}^{M} \sum_{t=1}^{T} \gamma_t(s, m) \log(b_{sm}(o_t)) ,$$

(2.14)

where $b_{sm}$ is the Gaussian pdf of mixture $m$ of state $s$; $o_t$ is the observation at time $t$; $\gamma_t(s, m)$ is the posterior probability of being at state $s$ and mixture $m$ at time $t$ given observation $o_t$; $S$ is the number of states in the HMM and $T$ is the duration of the observation, and $M$ is the number of Gaussian mixtures per HMM state.

The linear transformation and the bias are then applied to the Gaussian means. For simplicity, equation 2.13 is written as

$$\hat{\xi}_i = W_c \xi_i ,$$

(2.15)

where $\xi_i$ is defined as $[\mu_i', 1]'$ and $W_c$ is defined as $[A_c, b_c]$. Since

$$\log(b_{sm}(o_t)) = -\frac{1}{2}[d \log(2\pi) + \log |\Sigma_{sm}|]
- \frac{1}{2}(o_t - \mu_{sm})' \Sigma_{sm}^{-1}(o_t - \mu_{sm}) ,$$

(2.16)

and the transformations are only applied to means, we can ignore the first two terms. As a result, the partial derivatives of the $Q_b$ function w.r.t. $W_c$ becomes,

$$\frac{\partial Q_b}{\partial W_c} = \sum_{s=1}^{S} \sum_{m=1}^{M} \sum_{t=1}^{T} \gamma_t(s, m) \frac{\partial \log(b_{sm}(o_t))}{\partial W_c}
= -2 \sum_{i=1,i \in C}^{R} \sum_{t=1}^{T} \gamma_t(i) \Sigma^{-1}_i (o_t - W_c \xi_i) \xi_i' ,$$

(2.17)

where $R$ is the total number of Gaussians and $R = S \times M$, and $\gamma_t(i)$ is the posterior probability of being at the $i$th Gaussian at time $t$. By setting equation 2.17 to zero, we obtain

$$\sum_{i \in C} \sum_{t=1}^{T} \gamma_t(i) \Sigma^{-1}_i o_t \xi_i' = \sum_{i \in C} \sum_{t=1}^{T} \gamma_t(i) \Sigma^{-1}_i W_c \xi_i \xi_i'$$

(2.18)

and

$$Z = \sum_{i \in C} V_i \times W_c \times D_i ,$$

(2.19)
where

\[
Z = \sum_{i \in C} \sum_{t=1}^{T} \gamma_t(i) \Sigma_i^{-1} o_t \xi_i^t
\]

\[
V_i = \sum_{t=1}^{T} \gamma_t(i) \Sigma_i^{-1}
\]

\[
D_i = \xi_i \xi_i^t.
\]

\(W_c\) can be obtained by solving the simultaneous equations of equation 2.19. However, it is computationally expensive. If we assume that \(\Sigma_i\) is a diagonal matrix, then \(V_i\) is also diagonal. As a result, the \(q\)-th row of \([V_i \times W_c \times D_i] = (W_c)_q \times [(V_i)_{qq} \times D_i]\). Defining

\[
G_q = \sum_{i \in C} (V_i)_{qq} \times D_i,
\]

then

\[
(Z)_q = (W_c)_q \times G_q,
\]

\[
(W_c)_q = (Z)_q \times G_q^{-1}.
\]

It is possible that \(G_q\) is singular, if there is not enough adaptation data.

Note that \(W_c\) can be restricted to be a block diagonal transformation. The advantage of employing such a restriction is that it reduces the number of parameters to be estimated. As a result, it can speed up the adaptation process and occupy less space as well.

MLLR adaptation can also be applied to the covariances. However, it is known that the improvement is less significant compared with transforming the mean vectors [22]. Hence, MLLR was only applied to the mean vectors in our experiments.

### 2.4 Eigenspace-based MLLR (EMLLR) Speaker Adaptation

The traditional EV approach faces several technical problems. Firstly, the dimension of the supervector space is very high. In most cases, the number of speakers
is much smaller than the dimension of the supervector space ($N \ll D$), hence, the resulting eigenvoices span only a small portion of the supervector space and some noisy data may deteriorate the performance. Secondly, in the case of having multiple Gaussian mixtures per state, although one can apply the method proposed in [8] to handle the ordering problem of the Gaussian components, the dimension of the supervectors is huge and the computation of PCA and ML estimation becomes difficult. Thirdly, if each SD model is large, storing several eigenvoices during adaptation may be infeasible. These difficulties hinder the application of the EV approach. In this part, a variant of EV, known as eigenspace-based MLLR (EMLLR)[11] [13] [12] [3], is introduced and it can remedy the problems suggested above.

2.4.1 Parameter spaces of EMLLR

Similar to the EV approach, EMLLR involves 3 parameter spaces as follows,

- **Observation space**
  This is the acoustic feature space as mentioned in the section about EV adaptation. Compared with the supervector spaces, the dimension of this space is much smaller.

- **Supervector space**
  Instead of making an SD model for each training speaker, EMLLR applies MLLR to each training speaker with his training data. Then, the supervector space of EMLLR is formed by vectorizing each transformation matrix obtained from MLLR.

- **Eigenspace**
  PCA is performed similarly as the EV approach, and the resulting eigenmatrices are used to construct an MLLR transformation for the new speaker. Usually, we may extract a subset of the eigenvectors and the eigenspace is a subspace spanned by those eigenvectors.

2.4.2 Mechanism of EMLLR

Compared with the traditional EV approach, EMLLR has several advantages. Firstly, as shown in figure 2.4, MLLR allows the use of regression classes, so the
The dimension of the supervector space can be controlled. Secondly, the dimension of the supervector space is usually much smaller than the supervector space of the EV approach, so it requires less storage space during adaptation. The dimension of EMLLR’s supervector space can be calculated by

\[
\text{dim(supervector space)} = \text{no. of regression classes} \times \text{dim(observations space)} \\
\times (\text{dim(observations space)} + 1),
\]

which is generally much smaller than the EV’s supervector space. Thirdly, the output of EMLLR is just an MLLR transformation. Hence, EMLLR can be applied to multiple Gaussian mixtures models without the ordering problem in the EV approach.

Mathematically, suppose there is a set of \(N\) speaker-dependent (SD) acoustic models which are hidden Markov models (HMMs) of the same topology. These SD models are estimated from the speaker-independent (SI) model through MLLR transformation. For simplicity, the following discussion assumes that only one global MLLR transform is used; its extension to multiple MLLR transformations using regression classes of Gaussians should be straightforward. Thus, for the \(i\)th speaker, the mean vector of his \(r\)th Gaussian \(\mu_r^{(i)} \in \mathbb{R}^d\) is

\[
\mu_r^{(i)} = Y^{(i)'} \xi_r^{(SI)}
\]

Figure 2.4: Overview of EMLLR.
where $Y^{(i)'} \in \mathbb{R}^{d \times (d+1)}$ is the global MLLR transformation for the $i$th speaker, and $\xi^{(SI)}_r = [\mu^{(SI)}_r, 1]'$ is the augmented mean vector of the corresponding Gaussian of the SI model. A speaker transformation vector is obtained by vectorizing $Y$. (If we have multiple MLLR transformations, their vectorized matrices are stacked up to a speaker transformation supervector.) Let us denote $\text{vec}(Y)$ by $y$.

From the $N$ transformation vectors, $\{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\}$, PCA is performed, and the resulting eigenvectors are the vectorized eigenmatrices. The task of speaker adaptation is reduced to finding an MLLR transformation for the new speaker, which is assumed to lie in the span of the $M$ leading eigenmatrices (i.e. the $M$ eigenvectors with the largest eigenvalues). Thus, if a new speaker’s MLLR transformation is $Y$, then we have

$$\text{vec}(Y) = y = \sum_{m=1}^{M} w_m v_m,$$

(2.25)

where $w = [w_1, \ldots, w_M]'$ is the eigenmatrix weight vector, and $v_m$ is the $m$th vectorized eigenmatrix. Let $y = [y_1, y_2, \ldots, y_d]$ where $y_p \in \mathbb{R}^{d+1}$ is the $p$th row of $Y'$ (for $p = 1, \ldots, d$). Then $y_p$ is given by

$$y_p = \sum_{m=1}^{M} w_m v_{mp},$$

(2.26)

where $v_{mp}$ represents the $p$th row of the $m$th eigenmatrix.

Hence, the $r$th Gaussian mean of the new speaker model is

$$\mu_r = Y' \xi^{(SI)}_r$$

$$\Rightarrow \mu_{rp} = y' \xi^{(SI)}_r = \sum_{m=1}^{M} w_m (v'_{mp} \xi^{(SI)}_r),$$

(2.27)

where $\mu_{rp}$ is the $p$th component of $\mu_r$.

Given the adaptation data $O = \{o_1, o_2, \ldots, o_T\}$, the eigenmatrix weights can be estimated by maximizing the likelihood of $O$ as in the EV adaptation [26, 11]. Mathematically, one finds the optimal $\hat{w}$ by maximizing the following $Q(w)$ function:

$$Q(w) = - \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(o_t - \mu_r(w))' C_r^{-1}(o_t - \mu_r(w))$$

(2.28)
where \( \gamma_t(r) \) is the posterior probability of the observation sequence being at the \( r \)th Gaussian at time \( t \), and \( C_r \) is the covariance matrix of the \( r \)th Gaussian. Differentiating \( Q(w) \) w.r.t. each weight, \( w_m, m = 1, \ldots, M \), we get

\[
\frac{\partial Q(w)}{\partial w_m} = 2 \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(o_t - \mu_r(w))'C_r^{-1}\frac{\partial \mu_r(w)}{\partial w_m}.
\]

By setting the \( M \) derivatives to zero, the optimal weights are obtained by solving the system of \( M \) linear equations.

### 2.4.3 Robust EMLLR

Similar to the EV approach, the estimate found by EMLLR may not be reliable due to the small amount of adaptation data. The same parameter smoothing method is applied. The eigenmatrix weights are determined by solving a system of linear equations, and then the interpolation parameter, \( w_0 \), is found analytically.
CHAPTER 3

IMPROVING EIGENVOICE-BASED ADAPTATION BY KERNEL PCA

3.1 Kernel Methods

In the last decade, we witness a fast development in kernel-based learning methods. Some of the most popular methods including support vector machine (SVM) [46] [4] [9], kernel principal component analysis (KPCA) [5] and kernel fisher discriminant (KFD) [41] have shown practical usage in different areas.

The idea of kernel method is thoroughly discussed in [23], [6], [5] and [4]. One of the major advantages of kernel method is that, on the one hand, it enjoys a high dimensional representation of data and on the other hand, it avoids explicit mapping so the computation is efficient. The example below is borrowed from [5], and it illustrates why a high dimensional representation is beneficial.

In figure 3.1, the data points are 2-dimensional and they belong to two different classes. As observed, those two classes are not linearly separable. However, after applying a nonlinear mapping \( \varphi : (x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2) \) that maps the original 2d data to a 3d space, as shown in figure 3.1, the two classes become linearly separable.

Figure 3.1: An example from "Nonlinear Component Analysis as a Kernel Eigenvalue Problem".

21
While a high dimensional representation may be beneficial in classification, it also increases the computational complexity. From the above example, if the dimension of the observation space increases, the possible combination of dimensions increases exponentially as well. To tackle this problem, kernel methods compute the inner products in the feature space by using a kernel function $\varphi(x)'\varphi(y) = k(x, y)$. During the computation, the explicit form of the nonlinear map needs not be known and the computation does not involve the explicit mapping as well. As a result, it avoids expensive computation.

According to [46], a kernel function has to satisfy the following condition:

$$\forall f \in L_2(C): \int_C k(x, y)f(x)f(y)dxdy \geq 0,$$

(3.1)

where $C$ is some compact space and $C \subset \mathbb{R}^N$, $L_2(C)$ is a Hilbert space on $C$ and $k$ is a continuous kernel of a positive integral operator on $L_2(C)$. Some commonly used kernel functions include

- Gaussian kernel: $k(x, y) = exp(-\beta||x - y||^2)$
- Polynomial kernel: $((x \cdot y) + \theta)^d$
- Sigmoid kernel: $tanh(\kappa(x \cdot y) + \theta)$

### 3.1.1 Kernel principal component analysis (KPCA)

Kernel Principal Component Analysis (KPCA) can be considered as a nonlinear form of PCA. The nonlinearity is achieved by using kernel method. Theoretically, KPCA maps the data from the input space to a high dimensional feature space through some nonlinear map, and linear PCA is performed in that high dimensional space. During the process, the exact definition of the nonlinear map needs not be known, since kernel method allows the computation of inner product in the feature space by using the inner product in the input space and a suitable kernel function. To apply the kernel method, the PCA algorithm is rewritten in terms of inner products. Detailed description is available in [5] and in this section, a summary of the major steps will be shown.
In PCA, the centered covariance matrix is needed for eigendecomposition and it can be computed by the following formula

\[ \tilde{C} = HCH, \]  

(3.2)

where \( H = I - \frac{1}{N}11' \) and \( 1 = [1, 1, \ldots, 1]' \). Let \( \varphi \) be the mapping from the input space to the feature space, and \( \tilde{\varphi} \) be its centered version, and \( \sum_{i=1}^{N} \tilde{\varphi}(x_i) = 0 \). In the feature space, the centered covariance matrix is calculated by,

\[ \tilde{C} = \frac{1}{N} \sum_{i=1}^{N} \tilde{\varphi}(x_i)\tilde{\varphi}(x_i)' \]

\[ = \frac{1}{N} \tilde{\Phi}_\mathbf{x} \tilde{\Phi}_\mathbf{x}', \]  

(3.3)

where \( N \) is the number of training data points, \( x_i \) denotes the \( i \)th data point and \( \tilde{\Phi}_\mathbf{x} \) is defined as \([\tilde{\varphi}(x_1), \ldots, \tilde{\varphi}(x_N)]\). In [5], it is shown that all eigenvectors, \( \mathbf{v}_m \), lie in the span of \( \{\tilde{\varphi}(x_1), \ldots, \tilde{\varphi}(x_N)\} \). Therefore,

\[ \mathbf{v}_m = \sum_{i=1}^{N} \alpha_{mi}\tilde{\varphi}(x_i) \]

\[ = \tilde{\Phi}_\mathbf{x} \alpha_m. \]  

(3.4)

As we would like to perform PCA to extract principal components in the feature space, we have to solve the following eigenvalue problem,

\[ \tilde{C}\mathbf{v} = \lambda \mathbf{v} \]

\[ \Rightarrow \frac{1}{N} \tilde{\Phi}_\mathbf{x} \tilde{\Phi}_\mathbf{x}' \tilde{\Phi}_\mathbf{x} \alpha = \lambda \tilde{\Phi}_\mathbf{x} \alpha, \]  

(3.5)

where \( \lambda \) is the eigenvalue corresponding to \( \mathbf{v} \). Multiplying \( \tilde{\Phi}_\mathbf{x}' \) on both sides yields,

\[ \tilde{K} \alpha = N\lambda \alpha, \]  

(3.6)

where \( \tilde{K} = \tilde{\Phi}_\mathbf{x}' \tilde{\Phi}_\mathbf{x} \). It can be shown that the following equation yields all solutions of equation 3.6,

\[ \tilde{K} \alpha = N\lambda \alpha. \]  

(3.7)

Detailed description of this step is available in appendix A. After solving equation 3.7, we can express the \( m \)th eigenvector \( \mathbf{v}_m \) by,

\[ \mathbf{v}_m = \sum_{i=1}^{N} \frac{\alpha_m}{\sqrt{\lambda_m}} \tilde{\varphi}(x_i), \]  

(3.8)

\(^1\sim\) is used to indicate the centered version of a quantity.
where \( v_m \) in this equation is normalized to be an unit vector. The proof of the normalizing factor is also available in appendix A.

With proper choice of kernel function, KPCA can exploit possible nonlinear structure of the data. One drawback of KPCA is that as the eigenvectors exist in the feature space and their preimages may not exist in the input space, we can only compute inner products in the feature space through a kernel function which may be costly due to the kernel evaluation. However, by applying methods proposed in [29], we can still obtain an approximate preimage.

Another important advantage in KPCA is that if we choose a linear kernel, basically, KPCA reduces to PCA,

\[
\varphi(x) = x
\]

\[
k(x_i, x_j) = \varphi(x_i)'\varphi(x_j) = x_i'x_j .
\]

However, the difference is that in the original PCA, one has to perform eigendecomposition on the covariance matrix or the correlation matrix with dimension of \( \text{dim}(x) \times \text{dim}(x) \), but for KPCA, eigendecomposition is performed on the kernel matrix and its dimension is \( N \times N \). In case \( N \ll \text{dim}(x) \), performing eigendecomposition on the kernel matrix is more efficient. Although it may not be the case in general, eigenvoice approaches have this property. As a result, even a linear kernel, instead of other kernels, is chosen, KPCA still has advantages over the traditional PCA approach in eigenspace-based speaker adaptation.

### 3.2 Kernel Eigenvoice (KEV) Speaker Adaptation

Kernel eigenvoice speaker adaptation (KEV) is a nonlinear generalization of the traditional EV approach. In traditional EV, a new speaker is represented by a linear combination of eigenvoices, while in KEV, by incorporating KPCA instead of linear PCA, the eigenvectors (eigenvoices) are extracted in a nonlinear manner. As a result, the kernel eigenspace is expected to contain richer speaker information compared with the subspace used in the EV approach.

The basic idea of KEV is to map the supervectors to a high dimensional feature space as shown in figure 3.2, and then perform linear PCA in that space to obtain
eigenvoices. During the computation, the exact definition of the mapping needs not be known. It is achieved by using kernel method which allows us to compute inner products in the high dimensional feature space by choosing a suitable kernel function. The computation of kernel method depends only on the inner product in the input space and the kernel function.

3.2.1 Parameter spaces of KEV

As an extension of the original EV approach, the parameter spaces of KEV are similar to EV’s. Below are the descriptions of the KEV’s parameter spaces.

- Observation space
  This is the acoustic feature space as mentioned in the EV and EMLLR adaptation methods. Compared with other parameter spaces, this space has the smallest dimension.

- Supervector space
  Same as EV, a supervector is formed by concatenating the means in the HMM. KPCA is applied on this supervector space.

- Feature Space
  It is a high dimensional space mapped from the supervector space. This space is induced by the choice of kernel function and we do not know the
exact mapping between the supervector space and this feature space. Linear PCA is performed in this high dimensional feature space.

- Kernel Eigenspace
  
  After performing KPCA, we may extract a set of eigenvectors, which are called kernel eigenvoices, and kernel eigenspace is a space spanned by these eigenvectors.

### 3.2.2 Mechanism of KEV

As mentioned in section 3.1.1, an eigenvector in KPCA is expressed as

$$v_m = \sum_{i=1}^{N} \frac{\alpha_m}{\sqrt{\lambda_m}} \varphi(x_i) .$$

(3.9)

In the context of KEV, $x_i$ is the $i$th vectorized SD model, $N$ is the number of training speakers and $v_m$ is the $m$th kernel eigenvoice in the high dimensional feature space.

Since the exact definition of the nonlinear map, $\varphi$, is not known and the kernel method only allows computation of inner products in the feature space, the whole adaptation and decoding processes have to be expressed in terms of inner products. During the calculation of the expected log likelihood, we have to manipulate the distance between a Gaussian mean, $\mu_r$, and an observation, $o_t$, $\|\mu_r - o_t\|^2$. As the Gaussian mean, a part of the speaker supervector, is defined in the feature space, the observation frame has to be mapped to the feature space as well.

### 3.2.3 Composite kernel

However, a problem arises when we would like to calculate the distance between a segment of the supervector and an observation in the feature space, since if a single Gaussian kernel is used, we can only calculate the inner products between any two supervectors. A challenge in KEV is to preserve the constituent information in the nonlinear map.

A proposed solution in [28] is adopting a composite kernel. Since the Gaussian means are concatenated to form a supervector, we may map each Gaussian mean
with a separate kernel, \( k_r(\cdot, \cdot) \) and the inner product between two supervectors in the feature space is defined as,

\[
k(x_i, x_j) = G(k_r(x_{ir}, x_{jr})), r = 1, \ldots, R)
\]

(3.10)

where \( G \) is a function which combines the base kernel \( k_r \) into a valid composite kernel \( k \). Note that the composite kernel is valid if and only if it is symmetric and positive semidefinite [18]. There are many possible options for \( G \), but in [37], it is found that the direct sum composite kernel gives good performance. The definition of the direct sum kernel is as follows,

\[
k(x_i, x_j) = \sum_{r=1}^{R} k_r(x_{ir}, x_{jr})
\]

(3.11)

and a Gaussian kernel is chosen to be the base kernel \( k_r \). By using the direct sum composite kernel, the constituent information of a speaker supervector is preserved and as shown in the next section, we can compute the distance between a Gaussian mean and an observation frame in the feature space.

### 3.2.4 KEV adaptation

In general, KEV adaptation consists of three steps,

- Express the auxiliary function in terms of inner products.
- Initialize the weights of the kernel eigenvoices.
- Apply generalized expectation and maximization (GEM) algorithm for optimization.

Recall that the auxiliary function is

\[
Q_b(w) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(\mu_r - \mathbf{o}_t)'C_r^{-1}(\mu_r - \mathbf{o}_t).
\]

(3.12)

In KEV adaptation, the centered supervector of the new speaker, \( \tilde{\zeta} \), in the feature space is a linear combination of kernel eigenvoices, and it is expressed as

\[
\tilde{\zeta} = \sum_{m=1}^{M} w_m \mathbf{v}_m.
\]

(3.13)
By equation 3.9, it can be written as

$$
\tilde{\zeta} = \sum_{m=1}^{M} \sum_{n=1}^{N} \frac{w_m \alpha_{mn}}{\sqrt{\lambda_m}} \tilde{\varphi}(x_n) .
$$

(3.14)

To express the auxiliary function in terms of inner products, we have to choose a kernel function. In [28], it is discovered that the Gaussian kernel gives good performance. Therefore, in this section, Gaussian kernel is used to formulate the algorithm. A Gaussian kernel is defined as

$$
k_r(x_i, x_j) = \exp(-\beta_r \|x_i - x_j\|^2_{C_r}) ,
$$

(3.15)

where $\beta_r$ is a tunable parameter; $x_i$ and $x_j$ are two supervectors; $C_r$ is the covariance matrix of the $r$th Gaussian; $\|x_i - x_j\|^2_{C_r}$ denotes the squared distance between $x_i$ and $x_j$ normalized by $C_r$. Then, the inner product between an observation frame, $o_t$, and the corresponding constituent of the new speaker supervector, $\zeta_r$, in the feature space can be expressed as

$$
\zeta'_r \varphi_r(o_t) = \left[ \left( \sum_{m=1}^{M} \sum_{i=1}^{N} \frac{w_m \alpha_{mn}}{\sqrt{\lambda_m}} \varphi_r(\mu_i) \right) + \bar{\varphi}_r \right] \varphi_r(o_t)
$$

$$
= \left[ \left( \sum_{m=1}^{M} \sum_{i=1}^{N} \frac{w_m \alpha_{mn}}{\sqrt{\lambda_m}} \varphi_r(\mu_i) - \bar{\varphi}_r \right) + \bar{\varphi}_r \right] \varphi_r(o_t)
$$

$$
= \sum_{m=1}^{M} \sum_{i=1}^{N} \frac{w_m \alpha_{mn}}{\sqrt{\lambda_m}} (k_r(\mu_i, o_t) - \bar{\varphi}_r \varphi_r(o_t)) + \bar{\varphi}_r \varphi_r(o_t)
$$

$$
= A_t(r) + \sum_{m=1}^{M} w_m B_t(m, r) ,
$$

(3.16)

where,

$$
\bar{\varphi}_r = \frac{1}{N} \sum_{i=1}^{N} \varphi_r(\mu_{ir})
$$

(3.17)

$$
A_t(r) = \frac{1}{N} \sum_{i=1}^{N} k_r(\mu_{ir}, o_t)
$$

(3.18)

$$
B_t(m, r) = \sum_{i=1}^{N} \frac{\alpha_{ir}}{\sqrt{\lambda_m}} (k_r(\mu_{ir}, o_t) - A_t(r)) .
$$

(3.19)
Its derivative with respect to each eigenvoice weight $w_m$ is

$$
\frac{\partial k_r^{(kev)}}{\partial w_m} = B_t(m, r) .
$$

With equation 3.15 and 3.16, we can express the auxiliary function $Q_b(w)$ in terms of $k_r$ as

$$
Q_b(w) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(\|\mu_r - \omega_t\|^2) 
= \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r)(-\frac{1}{\beta_r} \log(A_t(r) + \sum_{m=1}^{M} B_t(m, r))) ,
$$

and the first derivative is

$$
\frac{\partial Q_b(w)}{\partial w_m} = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) \left( -\frac{B_t(m, r)}{\beta_r k_r^{(kev)}(\mu_r, \omega_t)} \right) .
$$

Due to the nonlinearity of the equations, there is no closed form solution for determining the values of $w$. Hence, gradient ascent method is applied to maximize the likelihood, and this requires the weights to be initialized before the first iteration of gradient ascent. SI model is considered as a reasonable initialization and it is projected to each eigenvoice to obtain the initial weights. This can be done by,

$$
w_m^{(SI)} = v'_m \hat{\varphi}(x^{(SI)}) 
= \sum_{i=1}^{N} \frac{\alpha_{mn}}{\sqrt{\lambda}} \hat{\varphi}(x_i) \hat{\varphi}(x^{(SI)}) 
= \sum_{i=1}^{N} \frac{\alpha_{mn}}{\sqrt{\lambda}} [\varphi(x_i) - \hat{\varphi}'][\varphi(x^{(SI)}) - \hat{\varphi}] 
= \sum_{i=1}^{N} \frac{\alpha_{mn}}{\sqrt{\lambda}} [k(x_i, x^{(SI)}) + \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} k(x_p, x_q) 
- \frac{1}{N} \sum_{p=1}^{N} (k(x_i, x_p) + k(x_p, x^{(SI)}))] .
$$

Since only a subset of eigenvectors are used and they do not span the whole space, there is reconstruction error, and the resulting model is not identical to the original SI model.
3.3 Speedup of KEV: Embedded KEV Adaptation (eKEV)

A problem of KEV is that it is substantially slower than the traditional EV approach. The reason is that the eigenvoices only exist in the feature space. So in order to compute the likelihood of an observation, one has to use kernel method to implicitly map the observation to the feature space, which is computationally expensive due to the kernel evaluation. In this section, a speedup version of KEV would be introduced and it is named embedded KEV adaptation (eKEV) [36].

The goal of eKEV is to speedup KEV by finding a preimage of the new speaker on the speaker supervector space from the feature space as shown in figure 3.3. By doing so, we can obtain an explicit speaker adapted model; hence, it avoids kernel evaluation during the calculation of the likelihoods; and it would greatly improve the efficiency.

![Diagram](image)

**Figure 3.3:** A preimage of a supervector in the feature space.

The preimage is obtained by using the method proposed in [29]. In which, the distances between the training speakers and the new speaker supervector in the feature space will be maintained in the input space in the least square sense.
3.3.1 eKEV adaptation

This section describes the adaptation procedure of eKEV and the procedure involves the following steps as mentioned in [36].

   The preimage algorithm in [29] uses Euclidean distance constraints, hence, the speaker supervectors are normalized by its own covariance.

   \[ z = C^{-\frac{1}{2}}x \]  

   where \( x \) is a supervector; \( C \) is the covariance matrix of those supervectors; \( z \) is the normalized supervector. In this section, \( s^{(ekev)}_z \) denotes the new speaker supervector in the normalized space and \( s^{(ekev)}_x \) denotes the supervector in the original space.

2. Similarity between the new speaker and the training speakers in the feature space.
   The similarity between the \( r \)th mean vector in the new speaker vector \( s^{(ekev)}_z \) and the \( r \)th mean vector in the \( j \)th training speaker vector \( x_j \) can be calculated by,

   \[ k_r(s^{(ekev)}_z, x_{jr}) = \phi_r(s^{(ekev)}_z)' \phi_r(z_{jr}) \equiv A_r(j) + \sum_{m=1}^{M} w_mB_r(m, j) \]  

   where

   \[ A_r(j) = \frac{1}{N} \sum_{i=1}^{N} k_r(z_{ir}, z_{jr}) \]  

   \[ B_r(m, j) = \sum_{i=1}^{N} \frac{\alpha_{mi}}{\sqrt{\lambda_m}} (k_r(z_{ir}, z_{jr}) - A_r(j)) \]  

3. Finding the distances between all training speakers and their centroid in the input space.
   Let the column vectors of \( Z = [z_1, \ldots, z_n] \) be the training speakers. Assuming that the rank of \( Z \) is \( q \), then by applying singular value decomposition (SVD) on the centered \( Z \), we have

   \[ \tilde{Z} = U\Lambda V' = UL \]  

   (3.28)
where $U = [e_1, \ldots, e_q]$ is an $n \times q$ matrix with orthonormal column vectors, 
$L$ is a $q \times q$ diagonal matrix containing the eigenvalues and 
$L = [l_1, \ldots, l_n]$ is a $q \times n$ matrix with column $l_i$ being the projections of $z_i$ onto the $e_j$’s. As a result, $\|l_i\|^2$ is the squared Euclidean distance between $z_i$ and the centroid of all training speakers. The distances will form the following vector,

$$d_0 = [\|l_1\|^2, \ldots, \|l_n\|^2]' \in \mathbb{R}^n.$$  \hfill (3.29)

4. Finding the distance constraints between the new speaker and the training speakers in the input space.

Assuming that the preimage $s^{(ekev)}_{zr}$ is in the span of all training speakers. The distances between the new speaker and the training speakers can be computed from equations 3.25, 3.26 and 3.27. If the direct sum composite kernel is used and each base kernel is a Gaussian kernel with the same $\beta$ (i.e. $\beta_r = \beta, \forall r$), then,

$$k_r(s^{(ekev)}_{zr}, z_{jr}) = \exp(-\beta\|s^{(ekev)}_{zr} - z_{jr}\|^2).$$  \hfill (3.30)

Therefore, the distance in the input space can be expressed as

$$d_{jr} = \|s^{(ekev)}_{zr} - z_{jr}\|^2 = -\frac{1}{\beta} \log k_r(s^{(ekev)}_{zr}, z_{jr}).$$  \hfill (3.31)

Then the squared Euclidean distances from the new speaker to all training speakers in the input space are collected into a $n$ dimensional vector,

$$d = [d_1, \ldots, d_n].$$  \hfill (3.32)

5. Finding the preimage.

From [29], an approximate preimage that optimally satisfies the distance constraints $d$ in the least square manner is given by,

$$s^{(ekev)}_{z} = -\frac{1}{2}U\Lambda^{-1}V'(d - d_0) + \bar{z}.$$  \hfill (3.33)

It can be written as

$$s^{(ekev)}_{z} = Pd + q,$$  \hfill (3.34)

where

$$P = -\frac{1}{2}U\Lambda^{-1}V'$$  \hfill (3.35)

$$q = -Pd_0 + \bar{z}.$$  \hfill (3.36)

Note that only $d$ depends on the weights of eigenvoices.
6. ML estimation of eigenvoices weights.

The new speaker model in the original space can be expressed as

\[ s_x^{(ekev)} = C^2 s_z^{(ekev)} = C^2 (Pd + q) . \]  

(3.37)

Hence, the \( r \)th mean vector in the supervector is

\[ s_{x_r}^{(ekev)} = C^2 (P_r d + q_r) , \]  

(3.38)

where suppose \( D \) is the dimension of the supervector space, \( P_r \in \mathbb{R}^{D \times n} \) consists of the \((r - 1)D + 1)\)th to \((rD)\)th rows of \( P \), and \( q_r = -P_r d_0 + z_r \).

With equation 3.38, we can perform the ML estimation by maximizing the following auxiliary function,

\[ Q(w) = -\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) \| o_t - s_{x_r}^{(ekev)}(w) \|^2_{C_r} . \]  

(3.39)

Differentiating it w.r.t. each eigenvoice weight, we have

\[ \frac{\partial Q}{\partial w_m} = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t(r) (o_t - s_{x_r}^{(ekev)}(w))' C_r^{-1} \frac{\partial s_{x_r}^{(ekev)}(w)}{\partial w_m} , \]  

(3.40)

where

\[ \frac{\partial s_{x_r}^{(ekev)}(w)}{\partial w_m} = C_r^2 \left( P_r \frac{\partial d(w)}{\partial w_m} \right) . \]  

(3.41)

Similar to the original KEV, there is no closed form solution for \( w \) due to the nonlinearity of the problem. Gradient Ascent method is applied to estimate the weights.

Note that for eKEV, instead of considering all training speakers in the distance constraint, it is possible to just consider some nearest neighbors. In such a case, there are fewer elements in the distance constraints.

### 3.4 Experimental Setup: Resource Management

In order to evaluate the effectiveness of EV, a simple experiment is conducted. This setup will be repeatedly used in this thesis to evaluate and compare different speaker adaptation methods.
DARPA Resource Management continuous speech database (RM1) was used to evaluate the effectiveness of various adaptation methods. RM1 comprises a speaker-independent (SI) section and a speaker-dependent (SD) section. The SI section consists of 3990 training utterances from 109 speakers. On the other hand, there are 12 speakers in the SD section, each having 600 utterances for training, 100 utterances for development, and 100 utterances for evaluation.

For acoustic modeling, forty-seven context-independent phoneme models were trained using the SI training set. Each phoneme model was a strictly left-to-right 3-state hidden Markov model (HMM) with a single Gaussian per state (multiple mixture models were used in later sections). In addition, there were a 1-state short pause model and a 3-state silence model. The acoustic vector has a dimension $d = 13$, consisting of 12 MFCCs and the normalized log energy extracted from speech frames of 25ms long at the frame rate of 100Hz.

Adaptation experiments were done with different amounts of adaptation data. Since the goal of this thesis is to improve rapid speaker adaptation, relatively small amount of adaptation data were used to evaluate the performance of different methods. In this thesis, 5-second and 10-second adaptation data were tested to evaluate the performance of different adaptation methods. For each of the 12 speakers in the SD section, 3 groups of adaptation data were randomly chosen from its 100 development utterances for each case, so that 5-second adaptation set was about 4 – 5s long, consisting of 2 – 3 utterances. Similarly, 10-second adaptation set was about 9 – 10s long, consisting of 3 – 4 utterances. For each adaptation method, the resulting adapted model for each speaker was tested on its 100 evaluation utterances using word-pair grammar. Reported results are averages across all groups in each adaptation set.

3.4.1 Experimental evaluation of EV

Instead of training an SD model for each speaker, MLLR with at most 32 regression classes was performed to produce SA models. SA models were then vectorized to perform PCA. The reason for using SA models instead of SD models is that RM1 does not have enough data for each speaker to train an SD model and several phonemes may not occur in the training data of some speakers. Correlation matrix was used to perform PCA instead of covariance matrix since it can
keep large scalar values from dominating the principal components. The silence and short pause models were removed before performing PCA since they do not represent any characteristics of a speaker. After creating a new speaker model, the silence and the short pause models were then copied to the new model from the SI model.

The performance of EV was first evaluated using different numbers of eigen-voices and the result is shown in figure 3.4.

From the result, it can be observed that the improvement is not very large. In general, more adaptation data yield better results. EV approach shows a performance drop when more eigenvoices were used. This shows that using the first few eigenvoices is enough for adaptation and this matches the finding in [26] where using the first few eigenvoices is adequate.

For the performance degradation when more eigenvoices were used, one possible reason is that there are not enough adaptation data to reliably estimate the eigenvoice weights. As shown in figure 3.4, the performance of EV using 5 seconds data degraded when 5 or more eigenvoices were used, while using 10 seconds data, the performance of using 5 eigenvoices is better than using 1 or 3 eigenvoices. When more eigenvoices were used, using both 5 seconds or 10 sec-
onds data experienced a performance drop. It may be due to the eigenvoices with small eigenvalues may not contain rich speaker information and possibly contain some noise.

Note that the conducted experiments are also restricted by the fact that SA models were used instead of SD models. As an SA model may not contain as much speaker information as an SD model, this may hamper the improvement of the EV approach.

3.4.2 Experimental evaluation of MLLR

A simple experiment was conducted to evaluate the performance of MLLR. The setup was the same as the EV experiment and the adaptation was performed using the HTK software version 3.2.1. Results are shown in table 3.1 where

Table 3.1: RM1: MLLR adaptation results with a single-mixture HMM.

<table>
<thead>
<tr>
<th>system</th>
<th>5 sec.</th>
<th>10 sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>67.02</td>
<td>67.02</td>
</tr>
<tr>
<td>MLLR-D</td>
<td>NA</td>
<td>68.48</td>
</tr>
<tr>
<td>MLLR-F</td>
<td>NA</td>
<td>73.58</td>
</tr>
</tbody>
</table>

MLLR-F denotes the MLLR adaptation with full transformation matrices and MLLR-D means the MLLR adaptation with diagonal transformation matrices.

From the results, we observe that:

1. MLLR does not work on 5-second adaptation set.

By default, HTK requires there are at least 700 samples of the adaptation data for each regression class, in order to guarantee that MLLR can reliably improve the adapted model. In our experiments, 700 samples means 7 seconds of speech data. As a result, for the 5-second adaptation set, the threshold had to be reduced to 300 samples in order to make sure that HTK would perform MLLR. However, the adaptation was very poor and we, instead, did not report the results. For both 5-second and 10-second adaptation sets, it turns out that only one global regression class was used in each case.
2. Given enough adaptation data, MLLR-F is better than MLLR-D.

From the results, it can be observed that MLLR performs well if more adaptation data is available. Again, the 5-second adaptation performance was very poor, no results were reported. In general, MLLR with full transformation achieves better performance than MLLR with diagonal transformation.

3. MLLR is better with 10 seconds of adaptation data and EV is better with 5 seconds of adaptation data.

With sufficient data (10 seconds), MLLR performs significantly better than the EV adaptation. For 5-second adaptation, although EV only performs slightly better than the SI model, MLLR fails to improve the accuracy due to the lack of data.

### 3.4.3 Experimental evaluation of EMLLR

A simple experiment was conducted to evaluate the performance of EMLLR. The setup was the same as the EV and MLLR experiments. Results are shown in 3.5.

Similar to the EV experiment, MLLR was performed on each training speaker to obtain 109 transformations. They were then vectorized to perform EMLLR.
adaptation. Only one regression class was adopted in the MLLR training for the sake of simplicity. The effect of multiple regression classes would be examined in later chapters.

From the results, it can be observed that EMLLR needs relatively more “eigen-voices” (eigenmatrices) compared with the EV approach. This finding matches the results in [13] where using more eigenmatrices is beneficial. In 5-second adaptation, EMLLR has slight improvement similar to the EV approach. In 10-second adaptation, there are enough adaptation data and EMLLR performs significantly better than the case that only 5 seconds adaptation data is available.

It can also be observed that if there are enough data, EMLLR performs better using more eigenmatrices. This behavior suggests that there is still useful speaker information captured in the eigenmatrices with small eigenvalues.

3.4.4 Experimental evaluation of eKEV

A simple experiment was conducted to evaluate the performance of eKEV. The setup was the same as the EV, MLLR and EMLLR experiments.

For the eKEV experiment, Gaussian kernel is used as the base kernel and direct sum composite kernel is chosen since they are found to have good performance in [28] and [37]. We followed the experiment setting in [35] which are repeated as follows:

- The initial eigenvoice weights were the projections of the SI model onto the corresponding kernel eigenvoices.
- \( \beta_r = \beta = 0.0005 \) for all Gaussians.
- The number of neighbours was 5.
- The neighbours were determined by likelihood.
- The learning rate of Gradient Ascent was 0.0001.

The results are shown in figure 3.6.

For some unknown reasons, eKEV has better performance on the 5-second adaptation data than the 10-second adaptation data. This phenomenon does not
Figure 3.6: Performance of eKEV with a single-mixture HMM.

occur using multiple-mixture model. From the results, it can be seen that eKEV has good performance on both 5-second set and 10-second set. The best number of eigenvoices is 10 but 7 eigenvoices give nearly identical result.

3.4.5 Comparison among EV, MLLR, EMLLR and eKEV

As shown in previous sections, several experiments were performed in order to study various adaptation methods including EV, MLLR, EMLLR and eKEV. Table 3.2 and table 3.3 are the short summaries of those experiments.

Table 3.2: Comparison of the best performance of different adaptation methods on a single-mixture HMM using 5 seconds of adaptation data.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Word accuracy</th>
<th>#Parameters</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>67.02%</td>
<td>NA</td>
<td>NA</td>
<td>67.26%</td>
<td>67.33%</td>
<td>72.42%</td>
</tr>
<tr>
<td>MLLR-D</td>
<td>67.02%</td>
<td>26</td>
<td>182</td>
<td>4</td>
<td>76</td>
<td>11</td>
</tr>
<tr>
<td>MLLR-F</td>
<td>67.26%</td>
<td>26</td>
<td>182</td>
<td>4</td>
<td>76</td>
<td>11</td>
</tr>
<tr>
<td>EV</td>
<td>67.26%</td>
<td>4</td>
<td>76</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMLLR</td>
<td>67.33%</td>
<td>4</td>
<td>76</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eKEV</td>
<td>72.42%</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results show that eKEV has the best performance in 5-second adaptation and MLLR-F is the best in 10-second adaptation.
Table 3.3: Comparison of the best performance of different adaptation methods on a single-mixture HMM using 10 seconds of adaptation data.

<table>
<thead>
<tr>
<th></th>
<th>SI</th>
<th>MLLR-D</th>
<th>MLLR-F</th>
<th>EV</th>
<th>EMLLR</th>
<th>eKEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word accuracy</td>
<td>67.02%</td>
<td>68.48%</td>
<td>73.58%</td>
<td>67.38%</td>
<td>70.52%</td>
<td>70.96%</td>
</tr>
<tr>
<td>#Parameters</td>
<td>-</td>
<td>26</td>
<td>182</td>
<td>6</td>
<td>110</td>
<td>11</td>
</tr>
<tr>
<td>Setting</td>
<td>-</td>
<td>1RC</td>
<td>1RC</td>
<td>5EV</td>
<td>109EV</td>
<td>10EV</td>
</tr>
</tbody>
</table>

For 5-second adaptation, the eigenspace-based methods achieve different degree of performance gain with the limited amount of data. Although MLLR-D also has very few parameters, in this task, it fails to improve the recognition accuracy. One of the possible reasons is that the MLLR-D framework only allows scaling and shifting for all mean vectors. Without rotation, MLLR-D may have difficulties in manipulating the relationship between the Gaussians if only a few regression classes are available. For the EV approach, the new speaker model (or transformation) is forced to lie in the span of some eigenvoices (or eigenmatrices). This constraint guarantees that the estimate remains in a reasonable area and it models speaker variation directly. Hence, it is a possible reason why eigenspace-based adaptation is more suitable for rapid speaker adaptation.

In our setting, eKEV shows promising performance in accomplishing rapid speaker adaptation. However, in real world application, multiple-mixture HMM is commonly used, so it is important to evaluate the recognition performance in such a case. For a quick glance, table 3.4 and table 3.5 show the performance of MLLR, EMLLR and eKEV when a 10-mixture HMM was used. More details about various adaptation methods on multiple-mixture HMM are discussed in chapter 5.

Table 3.4: Comparison of the best performance of different adaptation methods with a 10-mixture HMM using 5 seconds adaptation data.

<table>
<thead>
<tr>
<th></th>
<th>SI</th>
<th>MLLR-D</th>
<th>MLLR-F</th>
<th>EMLLR</th>
<th>eKEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word accuracy</td>
<td>78.27%</td>
<td>NA</td>
<td>NA</td>
<td>79.90%</td>
<td>77.01%</td>
</tr>
<tr>
<td>#Parameters</td>
<td>-</td>
<td>26</td>
<td>182</td>
<td>110</td>
<td>8</td>
</tr>
<tr>
<td>Setting</td>
<td>-</td>
<td>1RC</td>
<td>1RC</td>
<td>109EV</td>
<td>7EV</td>
</tr>
</tbody>
</table>

It can be observed that eKEV’s performance degraded when a multiple-mixture HMM was used, while MLLR and EMLLR still performed well. One
Table 3.5: Comparison of the best performance of different adaptation methods with a 10-mixture HMM using 10 seconds adaptation data.

<table>
<thead>
<tr>
<th></th>
<th>SI</th>
<th>MLLR-D</th>
<th>MLLR-F</th>
<th>EMLLR</th>
<th>eKEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word accuracy</td>
<td>78.27%</td>
<td>78.90%</td>
<td>82.15%</td>
<td>80.15%</td>
<td>78.63%</td>
</tr>
<tr>
<td>#Parameters</td>
<td>-</td>
<td>26</td>
<td>182</td>
<td>76</td>
<td>11</td>
</tr>
<tr>
<td>Setting</td>
<td>-</td>
<td>1RC</td>
<td>1RC</td>
<td>75EV</td>
<td>10EV</td>
</tr>
</tbody>
</table>

possible reason is that in eKEV adaptation, the distance constraint uses Euclidean distance. However, in a multiple-mixture HMM, different mixture has different mixture weights so they are of different importance, and this factor is ignored in the current framework of eKEV adaptation. As a result, the performance of eKEV degrades when a multiple-mixture model is used.

To achieve fast speaker adaptation, we have to find an adaptation method which performs well with multiple-mixture models. It should also perform consistently with a small amount of adaptation data. From all these experiments, we observe that exploiting nonlinearity can improve eigenspace-based adaptation methods, and also, EMLLR has satisfactory performance with a multiple-mixture HMM. An immediate question is if we can also kernelize EMLLR to improve EM-LLR’s performance, and this leads to the study of kernel eigenspace-based MLLR adaptation (KEMLLR).
CHAPTER 4

KERNEL EIGENSPACE-BASED MLLR (KEMLLR) ADAPTATION

4.1 Overview of KEMLLR

KEMLLR tries to improve KEV in a similar way of how EMLLR improves EV adaptation. Instead of using a speaker dependent (SD) model to represent a training speaker, KEMLLR uses an MLLR transformation as what EMLLR does. Also, KEMLLR incorporates nonlinearity as KEV does so that it provides a better generalization for eigenspace-based speaker adaptation. Basically, the design of KEMLLR is targeted at achieving the following goals:

- Good adaptation performance with small amount of data.
- Applicable on multiple-mixture model.
- Small amount of training data per training speaker.
- Flexible enough to cope with different tasks with different amount of training speakers, training data, and adaptation data.

This chapter will mainly discuss the mechanism of KEMLLR.

4.2 Parameter Spaces of KEMLLR

Below is a shortlist of the parameter spaces of KEMLLR. Some spaces appear in other eigenspace-based adaptation methods as well.

- Observation space
  
  This is the acoustic feature space as mentioned in other eigenspace-based adaptation methods. Compared with the supervector space or the feature space, this space has much smaller dimensionality.
• Transformation supervector space

Unlike EV or KEV, the supervector space of KEMLLR is formed by vectorizing the MLLR transformation matrices of the training speakers. Since KPCA can be applied separately on different regression classes or applied once on all regression matrices by concatenating all matrices into one single supervector, the dimension of this supervector space can be calculated by two formulas. If KPCA is applied separately on individual for each regression class, we have

\[
\text{dim(supervector space)} = \text{dim(observati} \text{on space)} + 1). \tag{4.1}
\]

On the other hand, if KPCA is applied jointly, we have

\[
\text{dim(supervector space)} = \text{no. of regression classes} \times \text{dim(observati} \text{on space)} + 1). \tag{4.2}
\]

• Feature space

It is a high dimensional space mapped from the supervector space. This space is induced by the choice of kernel function, but usually we do not know the exact mapping between the supervector space and this feature space. KPCA is equivalent to performing linear PCA in this high dimensional feature space.

• Kernel eigenspace

After performing PCA in the feature space, we may extract a set of eigenvectors and kernel eigenspace is a space spanned by those eigenvectors.

4.3 KEMLLR Adaptation

In KEMLLR adaptation, we try to improve EMLLR by exploiting the possible nonlinearity in the speaker transformation supervector space. This is achieved by replacing linear PCA by kernel PCA and the use of composite kernel.
4.3.1 Kernel eigenmatrices in the feature space

Let $k(\cdot, \cdot)$ be the kernel with an associated mapping $\varphi$ which maps a speaker’s transformation vector $y$ in the input supervector space to $\varphi(y)$ in the kernel-induced high dimensional feature space. Given the set of $N$ supervectors $\{y_1, \ldots, y_N\}$, their $\varphi$-mapped feature vectors are $\{\varphi(y_1), \ldots, \varphi(y_N)\}$. Let $\tilde{K}$ be the centered kernel matrix with $\tilde{K}_{ij} \equiv \tilde{k}(y_i, y_j) = \tilde{\varphi}(y_i)' \tilde{\varphi}(y_j)$ where $\tilde{\varphi}(y) = \varphi(y) - \bar{\varphi}$ and $\bar{\varphi} = \frac{1}{N} \sum_{i=1}^{N} \varphi(y_i)$.

As mentioned in section 3.1.1 of chapter 3, after performing KPCA, the centered supervector of the new speaker in the feature space $\tilde{\varphi^{(kemllr)}}(y)$ is given by

$$\tilde{\varphi^{(kemllr)}}(y) = \sum_{m=1}^{M} w_m v_m = \sum_{m=1}^{M} \sum_{i=1}^{N} \frac{w_m \alpha_{mi}}{\sqrt{\lambda_m}} \tilde{\varphi}(y_i) . \quad (4.3)$$

where $M$ is the number of eigenmatrices used; $v_m$ is the $m$th eigenmatrix; $\alpha_{mi}$ is the $i$th element of the $m$th eigenvector obtained by solving equation 3.7 and $\lambda_m$ is the $m$th eigenvalue obtained by solving equation 3.7.

4.3.2 Composite kernel

Analogous to the use of composite kernels to preserve the state information in kernel eigenvoice [37], the row information of each transformation matrix is preserved in KEMLLR using the direct sum composite kernel so that

$$k(y_i, y_j) = \sum_{h=1}^{C} \sum_{p=1}^{d} k_{hp}(y_{ihp}, y_{jhp}) , \quad (4.4)$$

where $y_{ihp}$ represents the part of $y_i$ corresponding to the $p$th row of the MLLR transformation matrix of the $h$th regression class before the $\varphi$-mapping.

Thus, the $\tilde{\varphi}_{hp}$-mapping of the $p$th row of the MLLR transform of the $h$th regression class for the new speaker’s supervector is given by

$$\tilde{\varphi}^{(kemllr)}_{hp}(y_{hp}) = \sum_{m=1}^{M} \sum_{i=1}^{N} \frac{w_m \alpha_{mi}}{\sqrt{\lambda_m}} \tilde{\varphi}_{hp}(y_{ihp}) . \quad (4.5)$$
4.3.3 Kernel evaluation

Let $\xi_r^{(SI)} = [\mu_r^{(SI)}, 1]'$ be the augmented mean vector. $\mu_r^{(SI)}$ is the mean vector of the $r$th Gaussian in the speaker independent (SI) model. Using equation (4.5), the similarity between $\varphi_{hp}^{(kemlr)}(y_{hp})$ and $\varphi_{hp}(\xi_r^{(SI)})$ can be computed as follows:

$$k_{hp}^{(kemlr)}(y_{hp}, \xi_r^{(SI)}) = \varphi_{hp}^{(kemlr)}(y_{hp})' \varphi_{hp}(\xi_r^{(SI)})$$

$$= A_{hp}(r) + \sum_{m=1}^{M} w_m B_{hp}(m, r), \quad (4.6)$$

where

$$A_{hp}(r) = \varphi_{hp}' \varphi_{hp}(\xi_r^{(SI)}) = \frac{1}{N} \sum_{i=1}^{N} k_{hp}(y_{ihp}, \xi_r^{(SI)}), \quad (4.7)$$

$$B_{hp}(m, r) = \sum_{i=1}^{N} \frac{\alpha_{mi}}{\sqrt{\lambda_m}} (k_{hp}(y_{ihp}, \xi_r^{(SI)}) - A_{hp}(r)), \quad (4.8)$$

and $\varphi_{hp} = \frac{1}{N} \sum_{i=1}^{N} \varphi_{hp}(y_{ihp})$. Notice that all the kernel values in equations (4.7,4.8) may be computed offline prior to adaptation.

Furthermore, the derivative of $k_{hp}^{(kemlr)}(y_{hp}, \xi_r^{(SI)})$ w.r.t. each eigenvoice weight $w_m$, $m = 1, \ldots, M$, is given by

$$\frac{\partial}{\partial w_m} \left( k_{hp}^{(kemlr)}(y_{hp}, \xi_r^{(SI)}) \right) = B_{hp}(m, r). \quad (4.9)$$

4.3.4 Gradient of Gaussian means

Equation (2.29) requires the gradient of $\mu_r^{(kemlr)}$ w.r.t. each eigenmatrix weight $w_m, m = 1, \ldots, M$. This can be obtained by using Gaussian kernels for the composite kernels,

$$k_{hp}(u, v) = \exp(-\beta_{hp} \|u - v\|^2),$$

and the identity $u'v = \frac{1}{2}(\|u\|^2 + \|v\|^2 - \|u - v\|^2)$. By letting $u = y_{hp}$ and $v = \xi_r^{(SI)}$, we have

$$\mu_{rp}^{(kemlr)} = \frac{1}{2} \left[ \|\xi_r^{(SI)}\|^2 + \frac{1}{\beta_{hp}} \log \left( \frac{k_{hp}^{(kemlr)}(y_{hp}, \xi_r^{(SI)})}{k_{hp}^{(kemlr)}(y_{hp}, 0)} \right) \right]. \quad (4.10)$$
Substituting equations (4.6,4.7,4.8) into equation (4.10), differentiating the result w.r.t. \(w_m\), and making use of the gradient in equation (4.9), we get
\[
\frac{\partial \mu_{ep}^{(kemlir)}}{\partial w_m} = \frac{1}{2\beta_{hp}} \left[ \frac{B_{hp}(m,r)}{k_{hp}^{(kemlir)}(y_{hp},\xi_{r}^{(SI)})} - \frac{B_{hp}(m,-1)}{k_{hp}^{(kemlir)}(y_{hp},0)} \right],
\]
where we use the index \(r = -1\) to represent a special augmented vector \(\xi_{-1}^{(SI)}\) which is the zero vector 0.

### 4.3.5 ML estimation of eigenmatrix weights by the quasi-Newton BFGS method

Using equation (4.11), the derivatives of \(Q(w)\) of equation (2.29) w.r.t. each of the \(M\) weights \(w_m, m = 1, \ldots, M\), can be obtained. However, Due to the non-linearity of the kernel functions, there is no closed form solution for the optimal \(w\). In the past [38], the weights are obtained by gradient ascent method and we notice that sometimes it is not effective and gets stuck. Now we replace it by the quasi-Newton BFGS optimization algorithm which consistently gives better solutions [39]. Quasi-Newton method is similar to the traditional Newton’s method and makes use of the Hessian to retrieve the Newton’s direction. However, it approximates the Hessian with an estimate that can be derived solely from the gradient. As a result, it is more efficient and it can enforce the Hessian estimate to be strictly positive-definite.

In the quasi-Newton method, the inverse of the Hessian matrix \(A^{-1}\) is approximated by \(H_i\) in an iterative procedure so that \(\lim_{i \to \infty} H_i = A^{-1}\), where \(H_i\) is the Hessian inverse in the \(i\)th iteration, and it has to be positive definite and symmetric. \(H_i\) is updated by the (BFGS) algorithm as follows:
\[
H_{i+1} = (I - \frac{s_i y_i'}{y_i' s_i})H_i(I - \frac{y_i s_i'}{y_i' s_i}) + \frac{s_i s_i'}{y_i' s_i},
\]
where
\[
s_i = w_{i+1} - w_i \quad \text{(4.13)}
\]
\[
y_i = \nabla Q(w_{i+1}) - \nabla Q(w_i) \quad \text{(4.14)}
\]
Detailed description and proof are available in [7].
Finally, the optimal eigenmatrix weights can be optimized iteratively by the following updating formula:

\[ w_{i+1} = -\lambda_i H_i \nabla Q(w)|_{w_i}, \]

where \( \lambda_i \) is the learning rate of the \( i \)th iteration to be determined by a line search algorithm, and the gradient can be computed from equations. (2.29,4.11).

4.3.6 Robust KEMLLR

Similar to KEV, when the amount of adaptation data is really small, the MLLR transformation found by KEMLLR may not be reliable. To get a more robust estimate, the transformation found by KEMLLR is interpolated with the identity matrix. Equivalently, a mean vector found by KEMLLR is interpolated with the corresponding SI mean vector as follows:

\[
\mu_{rp}^{(rkemllr)} = w_0 \mu_{rp}^{(SI)} + (1 - w_0) \mu_{rp}^{(kemllr)}, \quad 0 \leq w_0 \leq 1.0. \quad (4.15)
\]

And the gradients of the Gaussian means are updated as below:

\[
\frac{\partial \mu_{rp}^{(rkemllr)}}{\partial w_0} = \mu_{rp}^{(SI)} - \mu_{rp}^{(kemllr)}, \quad (4.16)
\]

and

\[
\frac{\partial \mu_{rp}^{(rkemllr)}}{\partial w_m} = (1 - w_0) \frac{\partial \mu_{rp}^{(kemllr)}}{\partial w_m}, \quad m = 1, \ldots, M. \quad (4.17)
\]

Then \( w_0 \) can then be co-optimized with other eigenvector weights by the BFGS quasi-Newton method.
CHAPTER 5

EXPERIMENTAL EVALUATION

The proposed KEMLLR speaker adaptation method was evaluated on the DARPA Resource Management continuous speech database RM1. The feature extraction was the same as the experiments conducted in section 3.4 of chapter 3, but the acoustic model was changed to a more accurate 10-mixture HMM.

5.1 Experimental Procedure

From the SI model, an SA model was constructed for each of the 109 speakers in the SI training set using the MLLR adaptation. As a result, we obtained a set of $N = 109$ transformation supervectors for deriving the kernel eigenmatrices. Experiments were performed with either 5s or 10s adaptation data. To improve reliability of the results, for each test speaker, 3 sets of adaptation data were randomly chosen from his 100 development utterances. All reported results are the averages of experiments over the 3 adaptation sets of all speakers, and the adapted models were tested on their 100 evaluation utterances using word-pair grammar.

The following models or adaptation methods are compared:

**SI:** speaker-independent model.

**MLLR-D:** MLLR adaptation with diagonal transformation.

**MLLR-F:** MLLR adaptation with full transformation.

**EMLLR:** robust eigenspace-based MLLR adaptation.

**eKEV:** robust embedded kernel eigenvoice adaptation.

**KEMLLR:** robust kernel EMLLR adaptation.
5.2 Finding the Kernel Parameter $\beta$

Since Gaussian kernel was chosen as the base kernel in formulating KEMLLR, we had to determine $\beta_{hp}$ which is a tunable parameter of the Gaussian kernel. For simplicity, we assume that $\forall h, p$, $\beta_{hp} = \beta$, so we have to choose a suitable value of $\beta$ for KEMLLR. This was done empirically, as shown in figure 5.1.

![Graph showing word recognition accuracy versus beta values.]

Figure 5.1: Finding a suitable value of $\beta$ for the Gaussian kernel in KEMLLR adaptation.

In the experiment, a 10-mixture HMM was used for acoustic modeling. KEMLLR with different values of $\beta$ was applied on a 10-second adaptation set sampled from the training speakers. Instead of testing all 109 speakers, 4 speakers were sampled and a 3-fold cross validation was performed with the available development data. As shown in the figure, KEMLLR has the best performance when $\beta = 0.001$ and this value was used in all KEMLLR experiments.

5.3 Effectiveness of BFGS quasi-Newton Method

Compared with EMLLR, KEMLLR cannot determine the eigenmatrix weights analytically and has to rely on some numerical algorithm. Hence, it is important to justify the effectiveness of the BFGS quasi-Newton method. To examine its effectiveness, EMLLR was performed in two different ways and their results were compared:
1. by solving a system of linear equations to determine the eigenmatrix weights and then the value of $w_0$ was determined analytically.

2. using BFGS quasi-Newton method to estimate the eigenmatrix weights and $w_0$ simultaneously.

Table 5.1: EMLLR results obtained by using BFGS and by solving a system of linear equations.

<table>
<thead>
<tr>
<th>system</th>
<th>SI</th>
<th>BFGS</th>
<th>Linear system</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-second</td>
<td>78.27%</td>
<td>79.37%</td>
<td>79.39%</td>
</tr>
<tr>
<td>10-second</td>
<td>78.27%</td>
<td>79.89%</td>
<td>79.89%</td>
</tr>
</tbody>
</table>

In the experiment, 50 eigenmatrices were used. From the result, it can be observed that the eigenmatrix weights estimated by BFGS quasi-Newton method yield nearly identical result compared with the result obtained by solving a system of linear equations. This suggests that the BFGS algorithm is very effective and it supports the use of this quasi-Newton algorithm in KEMLLR adaptation.

5.4 Performing KPCA(PCA) Separately or Simultaneously on Different Regression Classes

When EMLLR or KEMLLR are applied with multiple regression classes, they can perform PCA or KPCA separately on each class or on the concatenated transformation supervectors.

The advantage of performing KPCA (or PCA) on each class is that it allows a more flexible framework for estimating the transformation of the new speaker. However, a drawback of this approach is that it generally involves more parameters which may not be suitable if there are not enough data.

The advantage of performing KPCA (or PCA) simultaneously on all regression classes by concatenating transformation supervectors of different classes is that, it can exploit the possible correlation between different regression classes. However, the resulting supervector space may have very high dimension and the target transformation of the new speaker may not lie in the span of the limited number of training speakers.
Table 5.2: Performing KPCA(PCA) separately or simultaneously on 2 regression classes.

<table>
<thead>
<tr>
<th>Model/Adaptation</th>
<th>Word Accuracy 5s</th>
<th>Word Accuracy 10s</th>
<th>#eigenmatrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>78.27%</td>
<td>78.27%</td>
<td>-</td>
</tr>
<tr>
<td>EMLLR(sep)</td>
<td>79.89%</td>
<td>79.93%</td>
<td>75</td>
</tr>
<tr>
<td>EMLLR(sim)</td>
<td>78.79%</td>
<td>78.19%</td>
<td>75</td>
</tr>
<tr>
<td>KEMLLR(sep)</td>
<td>79.96%</td>
<td>81.79%</td>
<td>50</td>
</tr>
<tr>
<td>KEMLLR(sim)</td>
<td>79.61%</td>
<td>81.35%</td>
<td>75</td>
</tr>
</tbody>
</table>

In order to find out which scheme is better for rapid speaker adaptation, an experiment was carried out and the result is shown in table 5.2. In the experiment, a 10-mixture HMM was used for acoustic modeling. EMLLR and KEMLLR were tested using 2 regression classes. The number of eigenmatrices was found for each configuration to achieve the best performance. From the result, it can be observed that performing KPCA or PCA separately on each regression class is generally better. This finding matches the observation reported in [13] for EMLLR, and it appears that KEMLLR also has this property. In the rest of this chapter, all experiments that employed multiple regression classes, they were treated separately.

### 5.5 Number of Eigenmatrices and Regression Classes

Fig. 5.2 and Fig. 5.3 describe the complicated relationship among the number of eigenmatrices, number of regression classes, and the amount of data used in EMLLR or KEMLLR adaptation. Twenty-five, 50, 75, and 109 eigenmatrices were tried with one or two regression classes using 5s or 10s of adaptation speech. As expected, better adaptation performance results when more adaptation data are available. Notice that when EMLLR or KEMLLR are performed with multiple regression classes, they can perform PCA or KPCA separately on each class or on the concatenated transformation supervectors. In our investigation, the former always gave better adaptation performance than the latter one. As a consequence, all experiments reported here when multiple regression classes were used treated them separately. We have the following observations:
Figure 5.2: Performance of EMLLR adaptation with a 10-mixture HMM.

- On the one hand, more regression classes should give more detailed modeling and should give better results. On the other hand, more regression classes require more adaptation data as there are more weights to estimate. The effect is more pronounced for KEMLLR: with 2 regression classes, the performance actually drops with only 5s of adaptation speech, but is elevated when 10s of adaptation speech is provided.

- KEMLLR generally outperforms EMLLR adaptation when the same number of eigenmatrices and regression classes are employed using the same amount of adaptation speech. This shows that the leading eigenmatrices derived in KEMLLR using KPCA are more effective in capturing useful speaker information.

- When all eigenmatrices are employed, EMLLR adaptation performance may still improve. This may suggest that there are residual nonlinear information which cannot be covered by the leading eigenmatrices derived by PCA in EMLLR so that using all eigenmatrices may still improve the performance. However, this is not true for KEMLLR where using all kernel eigenmatrices will degrade the performance. That suggests that the trailing kernel eigenmatrices contain relatively small amount of speaker information, and most speaker specific information has been captured in the leading eigenmatrices. Thus, once again, KPCA helps to extract the non-
linear eigen-information more effective than PCA.

5.6 Comparison among Various Adaptation Methods

In this experiment, the SI model, MLLR-D, MLLR-F, EMLLR, eKEV and KEMLLR are compared at their best settings with a 10-mixture HMM. The results are summarized in Table 5.3. It is found that when only 5s of adaptation speech were available, even we lowered the threshold for MLLR-D and MLLR-F when we ran HTK, they still could not be run. eKEV is also not effective in 5-second set and the performance degraded when a multiple-mixture HMM was used. On the other hand, EMLLR successfully reduced the word error rate (WER) by 7.82%, and KEMLLR could reduce the WER by 11.4%. When 10s of adaptation speech were provided, KEMLLR can reduce WER by 17.3%. MLLR-F became effective in this setting and it matched the performance of KEMLLR with 17.63% WER reduction. EMLLR, eKEV and MLLR-D reduce WER by 8.47%, 1.66% and 2.90% respectively and again, they do not perform as well as KEMLLR.

The performance of eKEV with different number of eigenvoices is shown in figure 5.4. It appears that eKEV does not perform well with multiple-mixture HMM. One possible reason, as mentioned in chapter 3 section 3.4.4, is that the
Table 5.3: Performance of the SI model, MLLR, EMLLR, eKEV and KEMLLR adaptation using a 10-mixture HMM.

<table>
<thead>
<tr>
<th>Model/Adaptation</th>
<th>Word Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5s</td>
</tr>
<tr>
<td>SI</td>
<td>78.27%</td>
</tr>
<tr>
<td>eKEV</td>
<td>77.01%</td>
</tr>
<tr>
<td>MLLR-D</td>
<td>N/A</td>
</tr>
<tr>
<td>MLLR-F</td>
<td>N/A</td>
</tr>
<tr>
<td>EMLLR</td>
<td>79.97%</td>
</tr>
<tr>
<td>KEMLLR</td>
<td>80.75%</td>
</tr>
</tbody>
</table>

distance constraint currently ignores the mixture weights.

Figure 5.4: Performance of eKEV adaptation with a 10-mixture HMM.

The performance of KEMLLR with a single-mixture HMM was also evaluated. As shown in figure 5.5, KEMLLR has better performance on both 5-second and 10-second set compared with EMLLR. However, eKEV outperforms KEMLLR in the 5-second set. A possible reason is that KEMLLR has a larger amount of parameters compared with eKEV which is a disadvantage when the amount of data is very limited. When there are enough data (10-second), KEMLLR achieves similar performance compared with eKEV, but for 10-second set, MLLR-F has the best performance. Table 5.4 shows the overall results.
Figure 5.5: Performance of KEMLLR adaptation with a single-mixture HMM using different numbers of eigenmatrices.

Table 5.4: Adaptation performance of the SI model, MLLR, EMLLR, eKEV and KEMLLR adaptation with a single-mixture HMM.

<table>
<thead>
<tr>
<th>Model/Adaptation</th>
<th>Word Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5s</td>
</tr>
<tr>
<td>SI</td>
<td>67.02%</td>
</tr>
<tr>
<td>eKEV</td>
<td>72.42%</td>
</tr>
<tr>
<td>MLLR-D</td>
<td>N/A</td>
</tr>
<tr>
<td>MLLR-F</td>
<td>N/A</td>
</tr>
<tr>
<td>EMLLR(1RC)</td>
<td>67.33%</td>
</tr>
<tr>
<td>KEMLLR(1RC)</td>
<td>68.69%</td>
</tr>
</tbody>
</table>

5.7 Analysis of the Eigenmatrices

In eigenspace-based speaker adaptation, the extracted components are expected to represent certain inter-speaker variations such as accents, gender and age. Although successful modeling of any physical inter-speaker variations does not imply better recognition performance, such characteristics may help us better understand different adaptation methods. In this section, we try to analyze how EMLLR and KEMLLR represent inter-speaker variations.

In order to examine how EMLLR and KEMLLR model various types of inter-speaker variations, another corpus named TIDIGITS was used [34]. TIDIGITS
is a clean connected digit corpus. There are total 326 speakers and its standard training set and testing set each consists of 163 speakers. Each speaker has around 77 utterances with lengths ranging from one to seven digits. The ages of the speakers are from six to seventy, and the corpus divides the speakers into four groups, which are boy, girl, man and woman.

For acoustic modeling, digit models were trained using the training set. Each digit model was a strictly left-to-right 16-state HMM with a single Gaussian per state. In addition, there were a 1-state short pause model and a 3-state silence model. The acoustic vector has a dimension \( d = 13 \), consisting of 12 MFCCs and the normalized log energy extracted from speech frames of 25ms long at the frame rate of 100Hz.

For each testing speaker, 10 seconds of speech was sampled and used for EMLLR and KEMLLR adaptation. Multi-dimensional scaling (MDS) was applied to project the eigenmatrices weights speakers’ weight vectors into a 2-dimensional space in order to discover how EMLLR and KEMLLR modeled inter-speaker variations. Forty, 70, 100, 130 and 163 eigenmatrices were used for EMLLR and KEMLLR. For simplicity, both EMLLR and KEMLLR used one regression class only.

Figure 5.6 to figure 5.15 are the scatter plots of EMLLR and KEMLLR generated by MDS for adults only. These figures try to show how well EMLLR and KEMLLR use the first \( M \) eigenmatrices, where \( M = 40, \ldots, 163 \), to separate two genders. Children are removed in these two figures. It appears that KEMLLR can model gender information even with only 40 eigenmatrices. For EMLLR, it seems that EMLLR does not represent gender information when only a few eigenmatrices were available, but only when all eigenmatrices were used.

Figure 5.16 to figure 5.25 are the scatter plots of EMLLR and KEMLLR generated by MDS using all available speakers. In addition to checking how EMLLR and KEMLLR model gender information, these figures try to show if EMLLR and KEMLLR model other inter-speaker variations such as age group. From the figures, it appears that EMLLR did not separate the children and the women in the scatter plots. For KEMLLR, although there is still a great overlapping among children and women, we can still distinguish that there are two clusters each representing children and women respectively.
Figure 5.6: Scatter plot of EMLLR for adults using 40 eigenmatrices.

Figure 5.7: Scatter plot of KEMLLR for adults using 40 eigenmatrices.
Figure 5.8: Scatter plot of EMLLR for adults using 70 eigenmatrices.

Figure 5.9: Scatter plot of KEMLLR for adults using 70 eigenmatrices.
Figure 5.10: Scatter plot of EMLLR for adults using 100 eigenmatrices.

Figure 5.11: Scatter plot of KEMLLR for adults using 100 eigenmatrices.
Figure 5.12: Scatter plot of EMLLR for adults using 130 eigenmatrices.

Figure 5.13: Scatter plot of KEMLLR for adults using 130 eigenmatrices.
Figure 5.14: Scatter plot of EMLLR for adults using 163 eigenmatrices.

Figure 5.15: Scatter plot of KEMLLR for adults using 163 eigenmatrices.
Figure 5.16: Scatter plot of EMLLR for all speakers using 40 eigenmatrices.

Figure 5.17: Scatter plot of KEMLLR for all speakers using 40 eigenmatrices.
Figure 5.18: Scatter plot of EMLLR for all speakers using 70 eigenmatrices.

Figure 5.19: Scatter plot of KEMLLR for all speakers using 70 eigenmatrices.
Figure 5.20: Scatter plot of EMLLR for all speakers using 100 eigenmatrices.

Figure 5.21: Scatter plot of KEMLLR for all speakers using 100 eigenmatrices.
Figure 5.22: Scatter plot of EMLLR for all speakers using 130 eigenmatrices.

Figure 5.23: Scatter plot of KEMLLR for all speakers using 130 eigenmatrices.
Figure 5.24: Scatter plot of EMLLR for all speakers using 163 eigenmatrices.

Figure 5.25: Scatter plot of KEMLLR for all speakers using 163 eigenmatrices.
5.8 Significance Tests

In order to compare the various adaptation methods, it is important to identify if the performance difference among the systems are significant. A software developed by the National Institute of Standards and Technology (NIST) is used to conduct the significance tests. In the experiments, a two-tail 5% significant level is used. The results are shown in table C.1 and table C.2 in appendix C.

When there are 5 seconds data, table C.1 showed that KEMLLR outperformed other systems. EMLLR is the second best system, and the SI model is marginally better than eKEV. When there are 10 seconds data, table C.2 showed that KEMLLR and MLLR-F have similar performance and they outperformed other systems. EMLLR has similar performance of eKEV and it is better than MLLR-D and the SI model.

5.9 Time Complexity Analysis of Online Computation

This section discusses the time complexities of various adaptation methods including EV, MLLR, EMLLR, KEV, eKEV and KEMLLR. Below is a shortlist of the notations being used,

- $N$: The number of training speakers
- $R$: The number of Gaussians
- $T_1$: The length of the adaptation data
- $T_2$: The length of the testing data
- $M$: The number of parameters (for EMLLR and KEMLLR, $M$ is the number of parameters for each regression class.)
- $C$: The number of regression classes for MLLR, EMLLR and KEMLLR
- $D$: The dimension of the supervector space
- $d$: The dimension of the observation space
This section concentrates on the online computation. Therefore, steps which can be executed offline are ignored.

The time complexities provided have several underlying assumptions and below is the shortlist,

1. Diagonal covariances are used in the HMM.
2. Gaussian kernel is used as the base kernel for KEV, eKEV and KEMLLR.
3. Since $\gamma_t(r)$ has to be computed at the beginning of all adaptation methods, the analysis ignores the computation of this part.
4. The time complexity measures the number of floating point operations.
5. All unary or binary floating point operations including addition, multiplication, division and exponential function cost constant time.

With these assumptions, the following auxiliary function,

$$Q_b = \sum_{r=1}^{R} \sum_{t=1}^{T_2} \gamma_t(r)(\mathbf{o}_t - \mu_r)^T C_r^{-1}(\mathbf{o}_t - \mu_r),$$

(5.1)

costs $O(RT_2d)$ time. The reason is that $C$ is a diagonal matrix, so there are only $d$ multiplications related to $C$. Both $\mathbf{o}_t$ and $\mu_r$ are $d$-dimensional, as a result, the whole term,

$$(\mathbf{o}_t - \mu_r)^T C_r^{-1}(\mathbf{o}_t - \mu_r),$$

(5.2)

has $O(d)$ calculations and the calculations include multiplications and additions. In sum, there are $R$ Gaussians and $T_2$ frames of data, therefore, the auxiliary function has $O(RT_2d)$ calculations. By assumption 5, it is considered as having $O(RT_2d)$ complexity.

### 5.9.1 Time complexity of EV

The online computation of EV consists of three major steps:

1. Constructing a system of linear equations
2. Solving the linear system
3. Reconstructing the speaker adapted model

The construction of the linear system with \( M \) equations and \( M \) variables involves the computation of the gradients w.r.t. each parameter. Assuming that we can afford storing the values of \( e_{ki}C^{-1}e_{mr} \) before adaptation, constructing the system of linear equations costs \( O(RT_1Md) \) time. After the system is constructed, the parameters of EV are estimated analytically by solving the system of linear equations. Assuming the set of linear equations is not singular, if the system consists of \( M \) variables, a variant of Strassen’s algorithm can solve it at \( O(M^{\log_2(7)}) \) [43]. Although in practice, LU decomposition may be more efficient, using the variant of Strassen’s algorithm simplifies the complexity analysis. After the weights are determined, the reconstruction of the speaker adapted model involves a linear combination of the eigenvoices, so it costs \( O(MD) \).

Since \( M \ll N \) and \( D = R \times d \) for EV, the online time complexity is dominated by the time for constructing the linear system and hence, the complexity is \( O(RT_1Md) \).

5.9.2 Time complexity of EMLLR

Similarly, the online computation of EMLLR also consists of three major steps:

1. Construct a system of linear equations
2. Solve the linear system
3. Reconstruct the MLLR transformation

Let \( M \) be the number of eigenmatrices in use for each regression class. Constructing a system with \( M \) linear equations and \( M \) variables involves the computation of the gradients w.r.t. each parameter, and since EMLLR may have \( C \) regression classes, this implies EMLLR has \( C \) linear systems. The eigenmatrices of EMLLR has dimension of \( d \times (d + 1) \). Assuming that we have enough memory so that we can precompute the matrix products between the eigenmatrices and the Gaussians means of the SI model, then the complexity of constructing the system is similar to EV’s, which is \( O(RT_1Md) \). After the system is constructed, the eigenmatrix weights can be determined by Strassen’s algorithm and it costs
Finally, the reconstruction of the MLLR transformation needs $O(MD)$. Since $M \simeq N$ and $D = C \times d \times (d+1)$ for EMLLR, the online time complexity is $O(RT_1 Nd)$. If $N$ is very large and $M$ is still similar to $N$, then we cannot ignore the computation of step 2. In such a case, the complexity is $O(RT_1 Nd + CN\log_2(7))$.

### 5.9.3 Time complexity of MLLR

Equation 2.22 is an important step for MLLR. It computes the transformation for the target speaker. The computation of the matrix $Z$ costs $O(RT_1 d)$ and $G$ costs $O(Rd^2)$ since the matrix $D_t$ can be precomputed. Inversion of $G$ takes $O(d\log_2(7))$ time. Since we have $d$ rows and $C$ regression classes, as a result, computing the transformations need $O(RT_1 d^2 + Rd^2 + Cd\log_2(7)) = O(RT_1 d^2)$ time.

### 5.9.4 Time complexity of KEV

Unlike EV, EMLLR and MLLR, KEV is a nonlinear optimization problem and there is no closed form solution. Therefore, instead of discussing the overall time complexity, the effort for finding the gradients of the parameters is covered, which is required for Gradient Ascent or other iterative optimization algorithms.

The computation of each iteration of KEV consists of the following steps:

1. Compute $k_r(\mu_{ir}, o_t)$.
2. Cache all $A_t(r)$.
3. Cache all $B_t(m, r)$.
4. Compute the gradients as shown in equation 3.22.

Each kernel evaluation costs $O(d)$ time, so step 1 costs $O(RT_1 Nd)$. After the results have been stored, $A_t(r)$ and $B_t(m, r)$ can be computed at $O(RT_1 N)$ and $O(RT_1 MN)$ time respectively. The gradient w.r.t each eigenvoice weight can be calculated by equation 3.22 at $O(RT_1 M)$ time for each eigenvoice weight, therefore, it needs $O(RT_1 M^2)$ time in total. In sum, each iteration of KEV takes $O(RT_1 Nd + RT_1 N + RT_1 MN + RT_1 M^2) = O(RT_1 Nd)$ time since $M$ tends to be small in KEV adaptation.
However, unlike other adaptation methods, the speaker adapted model estimated by KEV only exists in the feature space. Therefore, the recognition process needs to be expressed in terms of inner products and implicitly map the observations to the feature space. The formula is similar to equation 3.21 but the observations are the testing data. The complexity is $O(RT_2Nd)$.

5.9.5 Time complexity of eKEV

eKEV, similar to KEV, is a nonlinear optimization problem and there is no closed form solution. Therefore, numerical algorithm is applied to estimate the eigenvoice weights. Instead of discussing eKEV’s overall complexity, the time complexity for finding the gradients of the parameters is covered. For simplicity, the number of neighbours, $n$, is assumed to be a small constant and log likelihood is used to determine the neighbours of the new speaker.

To determine the reference speakers, we have to compute the distances between the new speaker and the training speakers. If the reference speaker will not be changed during the optimization of eigenvoice weights, this part only computes once and its complexity is ignored. After the reference speakers are decided, we have to apply singular value decomposition (SVD). Since we are only interested in $n$ reference speakers, we may perform truncated SVD which can be computed efficiently since $n$ is small in our case, and we only need to compute the truncated SVD once for the new speaker. After these preprocessing steps, we can estimate the eigenvoice weights by using a numerical algorithm.

Equation 3.41 computes the gradient of the preimage of the speaker adapted model w.r.t. each eigenvoice weight. It involves the gradients of the distance constraints between the new speaker and the reference speakers in the input space. First of all, computing all $k_r$ costs $O(RMd)$ time since we have $R$ Gaussians, a $d$-dimensional observation space and both $A_r(j)$ and $B_r(m, j)$ can be computed offline. By assuming only a few neighbours are used to form the constraints, the computation of all distance constraints’ gradients costs $O(RMd)$ assuming $k_r$ and $B_r(m, j)$ have been cached. Similarly, the distance constraints take $O(Rd)$ time. Then, equation 3.40 costs $O(RT_1d)$ for each weight. If we can afford storing the values of $\sum_{t=1}^{T} \gamma_t(r)(o_t - \mu_r(w))'C_r^{-1}$, this step takes $O(RT_1d + RMd)$ time.

As a result, the online computation of eKEV costs $O(RT_1d + RMd + RMd +$
\(RMd + Rd + Rd) = O(RT_1d)\) time for each iteration since \(M\) tends to be small in eKEV adaptation.

### 5.9.6 Time complexity of KEMLLR

Similar to KEV and eKEV, KEMLLR is a nonlinear optimization problem and there is no closed form solution, so only the complexity for finding the gradients is covered.

The computation of each iteration of KEMLLR consists of the following steps:

1. Compute the similarity between the rows of transformation and the augmented means, \(k^{(\text{keMLLR})}_{hp}(y_{hp}, \xi_t^{(SI)})\).
2. Reconstruct the mean vectors.
3. Compute all \(\frac{\partial \mu_r^{(\text{keMLLR})}}{\partial w_m}\).
4. Compute the gradients as shown in equation 3.22.

Step one takes \(O(M)\) time assuming both \(A_{hp}(r)\) and \(B_{hp}(m, r)\) have been cached. Since we have \(R\) Gaussians and \(d\)-dimensional observation space, the total cost of step one is \(O(RMd)\). The reconstruction of all adapted mean vectors as shown in equation 4.10 costs \(O(Rd)\) time after we have cached the results of step one. Similarly, step three can be computed in \(O(Rd)\) time as well. The gradients can be computed by equation 2.29 as shown below

\[
\frac{\partial Q(w)}{\partial w_m} = 2 \sum_{r=1}^R \sum_{t=1}^T \gamma_t(r)(\omega_t - \mu_r(w))^\prime C^{-1}_r \frac{\partial \mu_r(w)}{\partial w_m}.
\] (5.3)

If we can afford storing the values of \(\sum_{t=1}^T \gamma_t(r)(\omega_t - \mu_r(w))^\prime C^{-1}_r\) first, step four takes \(O(RT_1d + RMd)\) time.

In sum, KEMLLR needs \(O(RMd + Rd + Rd + RT_1d + RMd) = O(RT_1d)\) time for each iteration since \(T_1\) is much larger than \(M\) in practice.

### 5.9.7 Summary of Time Complexity Analysis

In sum, EV, EMLLR and MLLR are faster than the adaptation algorithms rely on numerical methods including KEV, eKEV and KEMLLR. To compare different
approaches, the adaptation methods are divided into two groups.

The time complexity analysis shows that EMLLR is generally slower than EV and MLLR. However, if only a few eigenmatrices are used, then the computational effort of EMLLR is similar to EV’s, though it may not achieve its best adaptation performance. EV appears to be slightly faster than MLLR since $M$ tends to be smaller than $d$. However, this prediction may be crude since the analysis ignores many factors including, but not limited to, the time for reconstructing the SA model and the time for applying the transformations to the SI model. In addition, the analysis ignores offline computation which is expensive for EV and EMLLR, but MLLR does not need to perform offline computation before adaptation. Table 5.5 shows the time complexities of EV, MLLR and EMLLR.

KEV, eKEV and KEMLLR all rely on numerical algorithms, thus, it is necessary to assume that the rates of convergence of all three methods are similar in order to compare their complexities. Under this assumption, KEV is obviously slower than eKEV and KEMLLR. KEMLLR’s efficiency is similar to eKEV. However, in practice, KEMLLR is slower than eKEV due to the overhead generated by the larger amount of parameters and eigenmatrices. Table 5.6 shows the time complexities of KEV, eKEV and KEMLLR for each iteration.

Table 5.5: Time complexities of EV, MLLR and EMLLR.

<table>
<thead>
<tr>
<th>Adaptation</th>
<th>Overall complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV</td>
<td>$O(RT_1Md)$</td>
</tr>
<tr>
<td>MLLR</td>
<td>$O(RT_1d^2)$</td>
</tr>
<tr>
<td>EMLLR</td>
<td>$O(RT_1Nd)$</td>
</tr>
</tbody>
</table>

Table 5.6: Time complexities of KEV, eKEV and KEMLLR.

<table>
<thead>
<tr>
<th>Adaptation</th>
<th>Complexity of each iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEV</td>
<td>$O(RT_1Nd)$</td>
</tr>
<tr>
<td>eKEV</td>
<td>$O(RT_1d)$</td>
</tr>
<tr>
<td>KEMLLR</td>
<td>$O(RT_1d)$</td>
</tr>
</tbody>
</table>

73
CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Summary

In this thesis, eigenspace-based MLLR (EMLLR) speaker adaptation has been extended by incorporating KPCA, and the resulting adaptation method is named kernel eigenspace-based MLLR (KEMLLR). KEMLLR improves EMLLR by exploiting possible nonlinearity among different speakers. In the experiments, it was shown that KEMLLR achieves good performance with multiple-mixture HMM and this implies KEMLLR is applicable in various real world applications. KEMLLR was also compared with various adaptation methods including eigenvoice (EV), MLLR, EMLLR, kernel eigenvoice (KEV) and embedded kernel eigenvoice (eKEV) in order to study the strength and weakness of different adaptation methods.

The experiments were conducted on the DARPA Resource Management continuous speech database (RM1). In the experiments, it was found that when only 5s of adaptation speech were available, EMLLR successfully reduced the word error rate (WER) by 7.82%, and KEMLLR could reduce the WER by 11.40%. When 10s of adaptation speech were provided, KEMLLR can reduce WER by 17.30%. MLLR-F became effective in this setting and it matched the performance of KEMLLR with 17.63% WER reduction. EMLLR, eKEV and MLLR-D reduced WER by 8.47%, 1.66% and 2.90% respectively and again, they did not perform as well as KEMLLR.

6.2 Conclusion

The study of various adaptation methods including EV, MLLR, EMLLR, KEV, eKEV and KEMLLR shows that they may perform differently at different conditions. From the results of the experiments, it appears that if there are 10 seconds or more adaptation data, MLLR using full transformation gives the best performance. However, if the task is rapid speaker adaptation and there are less than 10
seconds of speech, it is beneficial to use either eKEV or KEMLLR depending on the chosen acoustic model. If the recognition system uses a single-mixture HMM, eKEV performs the best, but if a multiple-mixture HMM is used, KEMLLR gives the most promising performance.

The time complexity analysis shows that EV, MLLR and EMLLR are generally more efficient than other kernel based adaptation methods including KEV, eKEV and KEMLLR. The comparison of these adaptation methods in term of their online computation complexities can be summarized as follows:

\[
EV \simeq MLLR < EMLLR \ll eKEV \simeq KEMLLR \ll KEV
\]

By analyzing the eigenmatrix weights of EMLLR and KEMLLR when different numbers of eigenmatrices were used, it appears that KEMLLR can model gender information with very few eigenmatrices, while EMLLR does not represent gender information when only a few eigenmatrices were available. However, when all eigenmatrices were used, the result showed that EMLLR can separate male and female speakers. For other inter-speaker variation such as age group, EMLLR did not separate the children and the women in the scatter plots, but for KEMLLR, although there is still a great overlapping among children and women, we can still distinguish there are two clusters each representing children and women.

### 6.3 Contributions

The kernel eigenspace-based MLLR adaptation is proposed. It improves EMLLR by incorporating kernel principal component analysis (KPCA). The experiments showed that KEMLLR outperforms EMLLR. This suggests KEMLLR can exploit the possible nonlinearity among different speakers, and capture the speaker information successfully.

KEMLLR can handle multiple-mixture model and it provides promising performance in such a case. The experiments show eKEV may not perform as well as KEMLLR when a multiple-mixture model is used. This supports KEMLLR is applicable to more complicated recognition tasks, where multiple-mixture model is commonly used.

The Gaussian kernel and the direct-sum composite kernel are used in formulating KEMLLR. The direct-sum composite kernel can maintain the row information
which may be lost during the nonlinear mapping.

BFGS quasi-Newton method is employed to solve the optimization problem of KEMLLR. The algorithm can estimate the eigenmatrix weights effectively and efficiently with little additional computation compared with gradient ascent.

The time complexities of various adaptation methods including EV, MLLR, EMLLR, KEV, eKEV, and KEMLLR are analyzed under some mild assumptions. Comparisons are made and the analysis helps understand the efficiency of various methods.

The weight analysis of KEMLLR and EMLLR showed how these two adaptation methods model inter-speaker variations including gender and age group. The results help understand the behavior of KEMLLR and EMLLR.

6.4 Future work

Although KEMLLR has good performance, it is computationally more expensive than the traditional EV or EMLLR approach since it involves numerical optimization. To speed up KEMLLR, improving the efficiency of kernel computation may be beneficial since kernel evaluation is expensive during the computation of gradients. If Gaussian kernel is chosen as the base kernel, it may be worth considering some approximation methods to the exponential function. In [42], a fast approximation to exponential function is proposed which may help to speed up the kernel evaluation during the computation of the gradients.

In the experiments, only Gaussian kernel was used as the base kernel in formulating KEMLLR. Other kernels such as polynomial kernel and sigmoid kernel may also be investigated.

Currently, KEMLLR was applied on small vocabulary (TIDIGITS) and medium vocabulary (RM1) continuous speech recognition systems. In theory, KEMLLR can also be applied to large vocabulary systems, and it is worth to evaluate the performance of KEMLLR in such condition. However, due to the computational complexity, improving the efficiency of KEMLLR is an important challenge to be overcome.
REFERENCES


APPENDIX A

PROOFS FOR KPCA

A.1 Proof of centering of covariance matrix

Let \( N \) be the number of data points and \( x_i \) be the \( i \)th data point. \( \varphi \) is a nonlinear map induced by kernel and \( \tilde{\varphi} \) is a centered version which means \( \frac{1}{N} \sum_{i=1}^{N} \tilde{\varphi}(x_i) = 0 \). The centered kernel matrix, \( \tilde{K} \), can be calculated as follows:

\[
\tilde{K}_{ij} = \tilde{\varphi}(x_i)' \tilde{\varphi}(x_j) \\
= (\varphi(x_i) - \tilde{\varphi})(\varphi(x_j) - \tilde{\varphi}) \\
= \varphi(x_i)' \varphi(x_j) - \frac{1}{N} \sum_{q=1}^{N} \varphi(x_i)' \varphi(x_q) \\
- \frac{1}{N} \sum_{p=1}^{N} \varphi(x_p)' \varphi(x_j) + \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} \varphi(x_p)' \varphi(x_q) \\
= K_{ij} - \frac{1}{N} \sum_{q=1}^{N} K_{iq} - \frac{1}{N} \sum_{p=1}^{N} K_{pj} \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} K_{pq} \quad (A.1)
\]

where

\[
\tilde{\varphi} = \frac{1}{N} \sum_{i=1}^{N} \varphi(x_i) \quad (A.2)
\]

Therefore,

\[
\tilde{K} = HKH \quad (A.3)
\]

where \( H = I - \frac{1}{N} 11' \) and \( 1 = [1, 1, \ldots, 1]' \).

A.2 Proof of the normalizing factor used in KPCA

For each eigenvector \( v_m \), it is normalized by \( c_m \) as follows:

\[
v_m = \sum_{i=1}^{N} \frac{\alpha_{mi}}{c_m} \tilde{\varphi}(x_i) \quad (A.4)
\]
Since the eigenvectors are orthonormal to each other, for any eigenvector $\mathbf{v}_m$ in the feature space, we have

$$\mathbf{v}_m' \mathbf{v}_m = 1 \ .$$

(A.5)

Substituting equation A.4 into equation A.5, we get

$$c^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_m \alpha_{mj} \tilde{\varphi}(\mathbf{x}_i)' \tilde{\varphi}(\mathbf{x}_j)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_m \alpha_{mj} K_{ij}$$

$$= \mathbf{\alpha}_m'^t K \mathbf{\alpha}_m .$$

(A.6)

Since $K \mathbf{\alpha}_m = \lambda_m \mathbf{\alpha}_m$, therefore

$$c^2 = \mathbf{\alpha}_m'^t \lambda_m \mathbf{\alpha}_m$$

$$= \lambda_m (\mathbf{\alpha}_m'^t \mathbf{\alpha}_m)$$

$$= \lambda_m$$

$$c = \sqrt{\lambda_m} .$$

(A.7)

Hence,

$$\mathbf{v}_m = \sum_{i=1}^{N} \frac{\alpha_m}{\sqrt{\lambda_m}} \tilde{\varphi}(\mathbf{x}_i) .$$

(A.8)
APPENDIX B

IMPLEMENTATION ISSUES

B.1 Introduction to BLAS and LAPACK

Since KEMLLR uses numerical algorithm for optimization, it is generally slower than the traditional EV and EMLLR approaches which solve a system of linear equations for optimization. As a consequence, it is important to look into the details of the implementation in order to improve the efficiency.

From the formulation of KEMLLR, it can be observed that KEMLLR makes intensive use of matrix and vector operations. If one can improve the computational performance of those operations, it can be expected that KEMLLR can benefit a lot from it. Some existing libraries provide efficient matrix and vector operations. Basic Linear Algebra Subroutine (BLAS) and LAPACK [2] are among the most efficient and reliable libraries.

BLAS can be considered as an portable interface which provides basic vector and matrix operations. BLAS’s routines are divided into 3 levels. Level 1 BLAS does vector-vector operations, Level 2 BLAS performs matrix-vector operations and Level 3 BLAS is about matrix-matrix operations. There are various implementations of BLAS including, but not limited to, Intel Math Library and Automatically Tuned Linear Algebra Software (ATLAS) [47]. The experiments conducted in this thesis made intensive use of ATLAS library.

LAPACK is a high performance library for solving linear algebra problems including, but not limited to, simultaneous linear equations, least square solutions of linear systems, eigenvalue problems and singular value problems. LAPACK is written in FORTRAN, but there is also a version written in C language and this variant is known as CLAPACK.

Both ATLAS and LAPACK are being used in various products including MATLAB. In the experiments, the implementation of KEMLLR uses ATLAS and CLAPACK to handle matrix or vector operations. In the next section, a
simple experiment was conducted to evaluate the performance of ATLAS and CLAPACK.

### B.2 Experimental Evaluation

Since matrix multiplication plays an important role of most linear algebra operations, a simple experiment is conducted to evaluate the performance of matrix multiplication.

In the experiment, the products of two random real-valued square matrices with dimension $1024 \times 1024$ are computed for 100 times and the resulting time for computation is averaged. The efficiency of ATLAS is compared with a simple matrix multiplication algorithm which calculates the inner product of every row of the first matrix, and every column of the second matrix.

The experiment is conducted on a Pentium III-1GHz machine running Linux with kernel 2.4.18 and gcc version 3.3.2. The matrices have single word precision and SSE is enabled in the experiment. The version of ATLAS is 3.6.0, and table B.1 shows the average result.

Table B.1: Averaged time of computing multiplication of two $1024 \times 1024$ matrices.

<table>
<thead>
<tr>
<th>System</th>
<th>ATLAS</th>
<th>Simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1.82s</td>
<td>194.88s</td>
</tr>
</tbody>
</table>
APPENDIX C

SIGNIFICANCE TESTS

In the significance tests, various adaptation methods are compared. The abbreviations of the adaptation methods and the tests are summarized as follows:

SI: speaker-independent model.

MLLR-D: MLLR adaptation with diagonal transformation.

MLLR-F: MLLR adaptation with full transformation.

EMLLR: robust eigenspace-based MLLR adaptation.

eKEV: robust embedded kernel eigenvoice adaptation.

KEMLLR: robust kernel EMLLR adaptation.

MP: Matched Pair Sentence Segment (Word Error) Test.

SP: Signed Paired Comparison (Speaker Word Accuracy Rate) Test.

WI: Wilcoxon Signed Rank (Speaker Word Accuracy Rate) Test.

MN: McNemar (Sentence Error) Test.
Table C.1: Significance tests on 5-second adaptation data using a 10-mixture HMM.

<table>
<thead>
<tr>
<th></th>
<th>KEMLLR</th>
<th>EMLLR</th>
<th>eKEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>MP: KEMLLR</td>
<td>MP: EMLLR</td>
<td>MP: SI</td>
</tr>
<tr>
<td></td>
<td>SP: KEMLLR</td>
<td>SP: EMLLR</td>
<td>SP: same</td>
</tr>
<tr>
<td></td>
<td>WI: KEMLLR</td>
<td>WI: EMLLR</td>
<td>WI: SI</td>
</tr>
<tr>
<td></td>
<td>MN: KEMLLR</td>
<td>MN: same</td>
<td>MN: SI</td>
</tr>
<tr>
<td>KEMLLR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MP: KEMLLR</td>
<td>SP: KEMLLR</td>
<td>MP: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>SP: KEMLLR</td>
<td>WI: KEMLLR</td>
<td>SP: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>WI: KEMLLR</td>
<td>MN: KEMLLR</td>
<td>WI: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>MN: KEMLLR</td>
<td></td>
<td>MN: KEMLLR</td>
</tr>
<tr>
<td>EMLLR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MP: EMLLR</td>
<td>MP: EMLLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SP: EMLLR</td>
<td>SP: EMLLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WI: EMLLR</td>
<td>WI: EMLLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MN: EMLLR</td>
<td>MN: EMLLR</td>
</tr>
</tbody>
</table>
Table C.2: Significance tests on 10-second adaptation data using a 10-mixture HMM.

<table>
<thead>
<tr>
<th></th>
<th>KEMLLR</th>
<th>EMLLR</th>
<th>eKEV</th>
<th>MLLR-F</th>
<th>MLLR-D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SI</strong></td>
<td>MP: KEMLLR</td>
<td>MP: EMLLR</td>
<td>MP: same</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-D</td>
</tr>
<tr>
<td></td>
<td>SP: KEMLLR</td>
<td>SP: EMLLR</td>
<td>SP: same</td>
<td>SP: MLLR-F</td>
<td>SP: same</td>
</tr>
<tr>
<td></td>
<td>WI: KEMLLR</td>
<td>WI: EMLLR</td>
<td>WI: same</td>
<td>WI: MLLR-F</td>
<td>WI: same</td>
</tr>
<tr>
<td></td>
<td>MN: KEMLLR</td>
<td>MN: EMLLR</td>
<td>MN: same</td>
<td>MN: MLLR-F</td>
<td>MN: MLLR-D</td>
</tr>
<tr>
<td><strong>KEMLLR</strong></td>
<td>MP: KEMLLR</td>
<td>MP: KEMLLR</td>
<td>MP: same</td>
<td>MP: KEMLLR</td>
<td>MP: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>SP: KEMLLR</td>
<td>SP: KEMLLR</td>
<td>SP: same</td>
<td>SP: KEMLLR</td>
<td>SP: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>WI: KEMLLR</td>
<td>WI: KEMLLR</td>
<td>WI: same</td>
<td>WI: KEMLLR</td>
<td>WI: KEMLLR</td>
</tr>
<tr>
<td></td>
<td>MN: KEMLLR</td>
<td>MN: KEMLLR</td>
<td>MN: same</td>
<td>MN: KEMLLR</td>
<td>MN: KEMLLR</td>
</tr>
<tr>
<td><strong>EMLLR</strong></td>
<td>MP: EMLLR</td>
<td>MP: EMLLR</td>
<td>MP: MLLR-F</td>
<td>MP: EMLLR</td>
<td>MP: EMLLR</td>
</tr>
<tr>
<td></td>
<td>SP: EMLLR</td>
<td>SP: MLLR-F</td>
<td>SP: EMLLR</td>
<td>SP: EMLLR</td>
<td>SP: EMLLR</td>
</tr>
<tr>
<td></td>
<td>WI: EMLLR</td>
<td>WI: MLLR-F</td>
<td>WI: EMLLR</td>
<td>WI: EMLLR</td>
<td>WI: EMLLR</td>
</tr>
<tr>
<td><strong>eKEV</strong></td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
</tr>
<tr>
<td></td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
</tr>
<tr>
<td></td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
</tr>
<tr>
<td><strong>MLLR-F</strong></td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
<td>MP: MLLR-F</td>
</tr>
<tr>
<td></td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
<td>SP: MLLR-F</td>
</tr>
<tr>
<td></td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
<td>WI: MLLR-F</td>
</tr>
</tbody>
</table>